GRAPE: Generalizable Replaceable Atomic Position Engine Reference Manual

Generated by Doxygen 1.2.5

Fri Aug 24 17:08:33 2001

Contents

1 GRAPE: Generalizable Replaceable Atomic Position Engine Hierarchical Index **1**

1.1 GRAPE: Generalizable Replaceable Atomic Position Engine Class Hierarchy . . . 1

2 GRAPE: Generalizable Replaceable Atomic Position Engine Compound Index **3**

2.1 GRAPE: Generalizable Replaceable Atomic Position Engine Compound List . . . 3

3 GRAPE: Generalizable Replaceable Atomic Position Engine File Index **5**

3.1 GRAPE: Generalizable Replaceable Atomic Position Engine File List 5

4 GRAPE: Generalizable Replaceable Atomic Position Engine Class Documentation **9**

4.1 ABitSet Class Reference 9

4.2 AMatrix Class Reference 21

4.3 AMatrix::AMatrixRep Class Reference 36

4.4 AnimFrame Class Reference 39

4.5 Atom Class Reference 43

4.6 AtomBuildup Class Reference 50

4.7 AtomConstraint Class Reference 55

4.8 AtomRegistry Class Reference 61

4.9 base_iterator Class Reference 63

4.10 bf_iterator Class Reference 67

4.11 BinEnvelopeConstraint Class Reference 71

4.12 ConstIterator Class Reference 77

4.13 Constraint Class Reference 80

4.14 ConstraintManager Class Reference 97

4.15 ConstraintRegistry Class Reference 116

4.16 CycleAll Class Reference 119

4.17 ddEdge Class Template Reference 122

4.18 ddGraph Class Template Reference 125

4.19 ddVertex Class Template Reference 143
4.20 df_iterator Class Reference 150
4.21 DistanceConstraint Class Reference 154
4.22 DLList Class Template Reference 161
4.23 DLListNode Class Template Reference 174
4.24 DLListRep Class Template Reference 176
4.25 DSIterator Class Reference 178
4.26 DynString Class Reference 183
4.27 Envelope Class Reference 201
4.28 EnvelopeConstraint Class Reference 228
4.29 Grid Class Reference 241
4.30 gwNumerics Class Reference 261
4.31 gwRelaxAtom Class Reference 272
4.32 IdGen Class Reference 279
4.33 IntPriorityQueue Class Template Reference 283
4.34 IntVec Class Reference 296
4.35 Iterator Class Reference 305
4.36 ABitSet::Iterator Class Reference 308
4.37 jdErr_Eval Class Reference 311
4.38 jdIterator Class Reference 315
4.39 jdOcc_Count Class Reference 318
4.40 jdRand_Eval Class Reference 321
4.41 jVector Class Reference 325
4.42 KAdjMat Class Reference 342
4.43 Matrix Class Reference 345
4.44 Molecule Class Reference 356
4.45 neighbor_iterator Class Reference 374
4.46 path_iterator Class Reference 377
4.47 PositionConstraint Class Reference 385
4.48 Process Class Reference 391
4.49 Queue Class Template Reference 400
4.50 RanBaseClass Class Reference 403
4.51 RanChar Class Reference 406
4.52 RanDbl Class Reference 408
4.53 RanInt Class Reference 410
4.54 RanLetter Class Reference 412

4.55 RanWord Class Reference 414
4.56 RefRep Class Reference 416
4.57 RelaxSubset Class Reference 418
4.58 Residue Class Reference 422
4.59 ResidueRegistry Class Reference 435
4.60 SatisfyMolecule Class Reference 438
4.61 sdEdge Class Template Reference 441
4.62 sdGraph Class Template Reference 443
4.63 sdVertex Class Template Reference 471
4.64 Segment Class Reference 480
4.65 Segments Class Reference 483
4.66 Stack Class Template Reference 488
4.67 StopEval Class Reference 490
4.68 UnitWeight Class Reference 502
4.69 z Class Reference 505

5 GRAPE: Generalizable Replaceable Atomic Position Engine File Documentation 507

5.1 abbrev.h File Reference 507
5.2 abitset.C File Reference 509
5.3 abitset.h File Reference 512
5.4 ajiobin.h File Reference 513
5.5 amatrix.h File Reference 524
5.6 amatrix.inc File Reference 525
5.7 atom.h File Reference 526
5.8 atom_buildup.C File Reference 528
5.9 atom_buildup.h File Reference 529
5.10 atom_constraint.C File Reference 530
5.11 atom_constraint.h File Reference 531
5.12 atom_registry.C File Reference 532
5.13 atom_registry.h File Reference 533
5.14 bin_envelope_constraint.C File Reference 535
5.15 bin_envelope_constraint.h File Reference 536
5.16 bits.h File Reference 537
5.17 compoper.h File Reference 539
5.18 constraint.C File Reference 541
5.19 constraint.h File Reference 542

5.20 constraint_manager.C File Reference 544
5.21 constraint_manager.h File Reference 545
5.22 constraint_registry.C File Reference 547
5.23 constraint_registry.h File Reference 548
5.24 cycle_all.C File Reference 549
5.25 cycle_all.h File Reference 550
5.26 ddgraph.h File Reference 551
5.27 ddgraph.inc File Reference 555
5.28 ddvertex.h File Reference 557
5.29 ddvertex.inc File Reference 558
5.30 dens-trans.C File Reference 559
5.31 dist.C File Reference 561
5.32 distance_constraint.C File Reference 562
5.33 distance_constraint.h File Reference 563
5.34 dllist.h File Reference 564
5.35 dllist.inc File Reference 565
5.36 dlliter.inc File Reference 567
5.37 dsiter.inc File Reference 568
5.38 dynstring.C File Reference 569
5.39 dynstring.h File Reference 571
5.40 env_diff.C File Reference 573
5.41 envelope.C File Reference 574
5.42 envelope.h File Reference 575
5.43 envelope_constraint.C File Reference 577
5.44 envelope_constraint.h File Reference 578
5.45 err_eval.C File Reference 579
5.46 error_handler.h File Reference 585
5.47 globals.C File Reference 587
5.48 globals.h File Reference 588
5.49 grape.C File Reference 590
5.50 graph_weights.h File Reference 599
5.51 grid.C File Reference 600
5.52 grid.h File Reference 601
5.53 gw_numerics.C File Reference 603
5.54 gw_numerics.h File Reference 604
5.55 gw_relax_atom.C File Reference 605

5.56 gw_relax_atom.h File Reference 606
5.57 idgen.h File Reference 607
5.58 intvec.C File Reference 608
5.59 intvec.h File Reference 609
5.60 intvec.inc File Reference 610
5.61 ipqueue.h File Reference 613
5.62 ipqueue.inc File Reference 614
5.63 jd_anim_frame.C File Reference 615
5.64 jd_anim_frame.h File Reference 616
5.65 jd_err_eval.C File Reference 617
5.66 jd_err_eval.h File Reference 618
5.67 jd_iterator.C File Reference 619
5.68 jd_iterator.h File Reference 620
5.69 jd_occ_count.C File Reference 621
5.70 jd_occ_count.h File Reference 622
5.71 jd_rand_eval.C File Reference 623
5.72 jd_rand_eval.h File Reference 624
5.73 jd_relax_subset.C File Reference 625
5.74 jd_relax_subset.h File Reference 626
5.75 jd_stop_eval.C File Reference 627
5.76 jd_stop_eval.h File Reference 628
5.77 jsmisc.h File Reference 629
5.78 jvector.C File Reference 631
5.79 jvector.h File Reference 635
5.80 kadjmat.h File Reference 638
5.81 Makefile File Reference 639
5.82 Makefile File Reference 640
5.83 Makefile File Reference 641
5.84 Makefile File Reference 642
5.85 Makefile File Reference 643
5.86 matrix.C File Reference 644
5.87 matrix.h File Reference 645
5.88 molecule.C File Reference 647
5.89 molecule.h File Reference 648
5.90 molio.C File Reference 650
5.91 ocount.C File Reference 651

5.92 position_constraint.C File Reference 655
5.93 position_constraint.h File Reference 656
5.94 process.C File Reference 657
5.95 process.h File Reference 658
5.96 queue.h File Reference 659
5.97 rand_eval.C File Reference 660
5.98 randomgenerator.C File Reference 665
5.99 randomgenerator.h File Reference 666
5.100refrep.h File Reference 668
5.101residue.C File Reference 669
5.102residue.h File Reference 670
5.103residue_registry.h File Reference 672
5.104satisfy_molecule.h File Reference 674
5.105sdgalg.inc File Reference 675
5.106sdgiter.inc File Reference 676
5.107sdgraph.h File Reference 677
5.108sdgraph.inc File Reference 680
5.109sdvertex.h File Reference 681
5.110sdvertex.inc File Reference 682
5.111search-sort.h File Reference 683
5.112segments.inc File Reference 689
5.113stack.h File Reference 690
5.114streamoper.h File Reference 691
5.115top_process.h File Reference 692
5.116utils.C File Reference 693
5.117utils.h File Reference 694

Chapter 1

GRAPE: Generalizable Replaceable Atomic Position Engine Hierarchical Index

1.1 GRAPE: Generalizable Replaceable Atomic Position Engine Class Hierarchy

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

- ABitSet . . . 9
- AMatrix . . . 21
- Atom . . . 43
- AtomRegistry . . . 61
- base_iterator . . . 63
  - bf_iterator . . . 67
  - df_iterator . . . 150
  - neighbor_iterator . . . 374
- ConstIterator . . . 77
- Constraint . . . 80
  - AtomConstraint . . . 55
  - BinEnvelopeConstraint . . . 71
  - DistanceConstraint . . . 154
  - EnvelopeConstraint . . . 228
  - PositionConstraint . . . 385
- ConstraintManager . . . 97
- ConstraintRegistry . . . 116
- ddEdge . . . 122
- ddGraph . . . 125
- ddVertex . . . 143
- DLList . . . 161
  - Queue . . . 400
  - Stack . . . 488
- DLListNode . . . 174
- DSIterator . . . 178
- DynString . . . 183

Envelope . . . 201
Grid . . . 241
gwNumerics . . . 261
IdGen . . . 279
IntPriorityQueue . . . 283
IntVec . . . 296
Iterator . . . 305
ABitSet::Iterator . . . 308
jVector . . . 325
KAdjMat . . . 342
Matrix . . . 345
Molecule . . . 356
path_iterator . . . 377
Process . . . 391
SatisfyMolecule . . . 438
AnimFrame . . . 39
AtomBuildup . . . 50
CycleAll . . . 119
gwRelaxAtom . . . 272
jdErr_Eval . . . 311
jdIterator . . . 315
jdOcc_Count . . . 318
jdRand_Eval . . . 321
RelaxSubset . . . 418
StopEval . . . 490
RanBaseClass . . . 403
RanChar . . . 406
RanDbl . . . 408
RanInt . . . 410
RanLetter . . . 412
RanWord . . . 414
RefRep . . . 416
AMatrix::AMatrixRep . . . 36
DLListRep . . . 176
Residue . . . 422
ResidueRegistry . . . 435
sdEdge . . . 441
sdGraph . . . 443
sdVertex . . . 471
Segment . . . 480
Segments . . . 483
UnitWeight . . . 502
z . . . 505

Chapter 2

GRAPE: Generalizable Replaceable Atomic Position Engine Compound Index

2.1 GRAPE: Generalizable Replaceable Atomic Position Engine Compound List

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

ABitSet . . . 9
AMatrix . . . 21
AMatrix::AMatrixRep . . . 36
AnimFrame . . . 39
Atom . . . 43
AtomBuildup . . . 50
AtomConstraint . . . 55
AtomRegistry . . . 61
base_iterator . . . 63
bf_iterator . . . 67
BinEnvelopeConstraint . . . 71
ConstIterator . . . 77
Constraint . . . 80
ConstraintManager . . . 97
ConstraintRegistry . . . 116
CycleAll . . . 119
ddEdge . . . 122
ddGraph . . . 125
ddVertex . . . 143
df_iterator . . . 150
DistanceConstraint . . . 154
DLList . . . 161
DLListNode . . . 174
DLListRep . . . 176
DSIterator . . . 178
DynString . . . 183
Envelope . . . 201

EnvelopeConstraint . . . 228
Grid . . . 241
gwNumerics . . . 261
gwRelaxAtom . . . 272
IdGen . . . 279
IntPriorityQueue . . . 283
IntVec . . . 296
Iterator . . . 305
ABitSet::Iterator . . . 308
jdErr_Eval . . . 311
jdIterator . . . 315
jdOcc_Count . . . 318
jdRand_Eval . . . 321
jVector . . . 325
KAdjMat . . . 342
Matrix . . . 345
Molecule . . . 356
neighbor_iterator . . . 374
path_iterator . . . 377
PositionConstraint . . . 385
Process . . . 391
Queue . . . 400
RanBaseClass . . . 403
RanChar . . . 406
RanDbl . . . 408
RanInt . . . 410
RanLetter . . . 412
RanWord . . . 414
RefRep . . . 416
RelaxSubset . . . 418
Residue . . . 422
ResidueRegistry . . . 435
SatisfyMolecule . . . 438
sdEdge . . . 441
sdGraph . . . 443
sdVertex . . . 471
Segment . . . 480
Segments . . . 483
Stack . . . 488
StopEval . . . 490
UnitWeight . . . 502
z . . . 505

Chapter 3

GRAPE: Generalizable Replaceable Atomic Position Engine File Index

3.1 GRAPE: Generalizable Replaceable Atomic Position Engine File List

Here is a list of all files with brief descriptions:

abbrev.h . . . 507
abitset.C . . . 509
abitset.h . . . 512
ajiobin.h . . . 513
amatrix.h . . . 524
amatrix.inc . . . 525
atom.h . . . 526
atom_buildup.C . . . 528
atom_buildup.h . . . 529
atom_constraint.C . . . 530
atom_constraint.h . . . 531
atom_registry.C . . . 532
atom_registry.h . . . 533
bin_envelope_constraint.C . . . 535
bin_envelope_constraint.h . . . 536
bits.h . . . 537
compoper.h . . . 539
constraint.C . . . 541
constraint.h . . . 542
constraint_manager.C . . . 544
constraint_manager.h . . . 545
constraint_registry.C . . . 547
constraint_registry.h . . . 548
cycle_all.C . . . 549
cycle_all.h . . . 550
ddgraph.h . . . 551
ddgraph.inc . . . 555

ddvertex.h . . . 557
ddvertex.inc . . . 558
dens-trans.C . . . 559
dist.C . . . 561
distance_constraint.C . . . 562
distance_constraint.h . . . 563
dllist.h . . . 564
dllist.inc . . . 565
dlliter.inc . . . 567
dsiter.inc . . . 568
dynstring.C . . . 569
dynstring.h . . . 571
env_diff.C . . . 573
envelope.C . . . 574
envelope.h . . . 575
envelope_constraint.C . . . 577
envelope_constraint.h . . . 578
err_eval.C . . . 579
error_handler.h . . . 585
globals.C . . . 587
globals.h . . . 588
grape.C . . . 590
graph_weights.h . . . 599
grid.C . . . 600
grid.h . . . 601
gw_numerics.C . . . 603
gw_numerics.h . . . 604
gw_relax_atom.C . . . 605
gw_relax_atom.h . . . 606
idgen.h . . . 607
intvec.C . . . 608
intvec.h . . . 609
intvec.inc . . . 610
ipqueue.h . . . 613
ipqueue.inc . . . 614
jd_anim_frame.C . . . 615
jd_anim_frame.h . . . 616
jd_err_eval.C . . . 617
jd_err_eval.h . . . 618
jd_iterator.C . . . 619
jd_iterator.h . . . 620
jd_occ_count.C . . . 621
jd_occ_count.h . . . 622
jd_rand_eval.C . . . 623
jd_rand_eval.h . . . 624
jd_relax_subset.C . . . 625
jd_relax_subset.h . . . 626
jd_stop_eval.C . . . 627
jd_stop_eval.h . . . 628
jsmisc.h . . . 629
jvector.C . . . 631
jvector.h . . . 635
kadjmat.h . . . 638
LeakTracer/Makefile . . . 639

lib/Makefile . . . 640
Makefile . . . 641
src/aux/Makefile . . . 642
src/Makefile . . . 643
matrix.C . . . 644
matrix.h . . . 645
molecule.C . . . 647
molecule.h . . . 648
molio.C . . . 650
ocount.C . . . 651
position_constraint.C . . . 655
position_constraint.h . . . 656
process.C . . . 657
process.h . . . 658
queue.h . . . 659
rand_eval.C . . . 660
randomgenerator.C . . . 665
randomgenerator.h . . . 666
refrep.h . . . 668
residue.C . . . 669
residue.h . . . 670
residue_registry.h . . . 672
satisfy_molecule.h . . . 674
sdgalg.inc . . . 675
sdgiter.inc . . . 676
sdgraph.h . . . 677
sdgraph.inc . . . 680
sdvertex.h . . . 681
sdvertex.inc . . . 682
search-sort.h . . . 683
segments.inc . . . 689
stack.h . . . 690
streamoper.h . . . 691
top_process.h . . . 692
utils.C . . . 693
utils.h . . . 694

Chapter 4

GRAPE: Generalizable Replaceable Atomic Position Engine Class Documentation

4.1 ABitSet Class Reference

`#include <abitset.h>`

Public Methods

- **ABitSet** ()
- **ABitSet** (**uint**)
- **ABitSet** (const ABitSet &)
- **~ABitSet** ()
- const ABitSet& **operator=** (const ABitSet &)
- bool **Resize** (**uint** n)
- bool **IsAllSet** () const
- bool **IsAllClear** () const
- bool **IsEmpty** () const
- bool **operator[]** (**uint**) const
- bool **IsSet** (**uint** i) const
- void **Set** (**uint**)
- void **Clear** (**uint**)
- void **Invert** (**uint**)
- bool **In** (**uint** i) const
- void **Add** (**uint** i)
- void **Remove** (**uint** i)
- void **Set** (**uint**, **uint**)
- void **Clear** (**uint**, **uint**)
- void **Invert** (**uint**, **uint**)
- void **SetAll** ()
- void **ClearAll** ()
- void **InvertAll** ()

- const ABitSet& **operator|=** (const ABitSet &)
- const ABitSet& **operator &=** (const ABitSet &)
- const ABitSet& **operator^=** (const ABitSet &)
- const ABitSet& **operator-=** (const ABitSet &)
- ABitSet **operator|** (const ABitSet &) const
- ABitSet **operator &** (const ABitSet &) const
- ABitSet **operator^** (const ABitSet &) const
- ABitSet **operator-** (const ABitSet &) const
- bool **Increment** ()
- bool **Decrement** ()
- int **Compare** (const ABitSet &) const
- int **Number** () const
- int **Count** () const
- **MAKECOMPARISONS_WITH_COMPARE** (ABitSet)

Static Public Methods

- **uint DefaultSize** (**uint** s)
- **uint DefaultSize** ()

Static Protected Attributes

- **uint default_size** = 256

Private Attributes

- **uint num**
- char* **data**

Friends

- ostream& **operator<<** (ostream &os, const ABitSet &)
- ostream& **OutputBinary** (ostream &, const ABitSet &)
- istream& **InputBinary** (istream &, ABitSet &)
- ostream& **operator<=** (ostream &os, const ABitSet &bs)
- istream& **operator>=** (istream &is, ABitSet &bs)

4.1.1 Constructor & Destructor Documentation

4.1.1.1 ABitSet::ABitSet ()

Definition at line 28 of file abitset.C.

```
00029 : num(default_size)
00030 {
00031         data = new char[b2c1(num)];
00032         ClearAll();
00033
```

4.1.1.2 ABitSet::ABitSet (uint *n*)

Definition at line 35 of file abitset.C.

```
00036 : num(n)
00037 {
00038         if (num<1)
00039                 num = 1;
00040         data = new char[b2c1(num)];
00041         ClearAll();
00042
```

4.1.1.3 ABitSet::ABitSet (const ABitSet & *abs*)

Definition at line 44 of file abitset.C.

```
00045 : num(abs.num)
00046 {
00047         const uint x=b2c1(num);
00048         data = new char[x];
00049         memcpy(data, abs.data, x);
00050
```

4.1.1.4 ABitSet::~ABitSet ()

Definition at line 52 of file abitset.C.

```
00053 {
00054         delete[] data;
00055         data = 0;
00056         num = 0;
00057
```

4.1.2 Member Function Documentation

4.1.2.1 void ABitSet::Add (uint *i*) [inline]

Definition at line 75 of file abitset.h.

```
00076                                 { Set(i); }
00077
```

4.1.2.2 void ABitSet::Clear (uint *i*, uint *j*)

Definition at line 163 of file abitset.C.

```
00164 {
00165         if (i>j)
00166                 util_swap(i,j);
00167         if ( j>=num )
00168                 return;
00169
00170         for (uint k=i; k<=j; ++k)
00171                 Clear(k);
00172
```

4.1.2.3 void ABitSet::Clear (uint *i*)

Definition at line 133 of file abitset.C.

Referenced by Clear(), KAdjMat::Disconnect(), and Remove().

```
00134 {
00135         if ( i>=num )
00136                 return;
00137
00138         data[b2c0(i)] &= ~b2i(i);
00139
```

4.1.2.4 void ABitSet::ClearAll ()

Definition at line 215 of file abitset.C.

Referenced by ABitSet(), bf_iterator::reset(), and df_iterator::reset().

```
00216 {
00217         memset(data, 0, b2c1(num));
00218
```

4.1.2.5 int ABitSet::Compare (const ABitSet & *abs*) const

Definition at line 350 of file abitset.C.

```
00351 {
00352         if (num != abs.num)
00353                 return abs.num - num;
00354
00355         uint    n = b2c1(num);
00356
00357         for (uint i=0; i<n; ++i)
00358         {
00359                 if (data[i] != abs.data[i])
00360                         return abs.data[i] - data[i];
00361         }
00362
```

4.1.2.6 int ABitSet::Count () const

Definition at line 229 of file abitset.C.

```
00230 {
00231         uint    n = b2c1(num);
00232         int     tot = 0;
00233         for (uint i=0; i<n; ++i)
00234                 tot += bit_count(data[i]);
00235         return tot;
00236
```

4.1.2.7 bool ABitSet::Decrement ()

Definition at line 330 of file abitset.C.

```
00331 {
00332         uint    n = b2c1(num);
00333         bool            carry_flag = 1;
00334         uint    i = 0;
00335
00336         while ( carry_flag && (i<n) )
00337         {
00338                 carry_flag = 0;
00339                 int     z = data[i];
00340                 ++z;
00341                 if (z & 256)
00342                         carry_flag = 1;
00343                 data[i] = z & 255;
00344                 ++i;
00345         }
00346
```

4.1.2.8 uint ABitSet::DefaultSize () [static]

Definition at line 109 of file abitset.C.

```
00110 {
00111         return default_size;
00112
```

4.1.2.9 uint ABitSet::DefaultSize (uint *s*) [static]

Definition at line 99 of file abitset.C.

```
00100 {
00101         uint    o = default_size;
00102         if (s<1)
00103                 s = 1;
00104         default_size = s;
00105         return o;
00106
```

4.1.2.10 bool ABitSet::In (uint *i*) const [inline]

Definition at line 73 of file abitset.h.

```
00074                                 { return ($\ast$this)[i]; }
00075
```

4.1.2.11 bool ABitSet::Increment ()

Definition at line 310 of file abitset.C.

```
00311 {
00312         uint    n = b2c1(num);
00313         bool            carry_flag = 1;
00314         uint    i = 0;
00315
00316         while ( carry_flag && (i<n) )
00317         {
```

```
00318                 carry_flag = 0;
00319                 int     z = data[i];
00320                 ++z;
00321                 if (z & 256)
00322                         carry_flag = 1;
00323                 data[i] = z & 255;
00324                 ++i;
00325         }
00326
```

4.1.2.12 void ABitSet::Invert (uint *i*, uint *j*)

Definition at line 175 of file abitset.C.

```
00176 {
00177         if (i>j)
00178                 util_swap(i,j);
00179         if ( j>=num )
00180                 return;
00181
00182         for (uint k=i; k<=j; ++k)
00183                 Invert(k);
00184
```

4.1.2.13 void ABitSet::Invert (uint *i*)

Definition at line 142 of file abitset.C.

Referenced by Invert().

```
00143 {
00144         if ( i>=num )
00145                 return;
00146
00147         data[b2c0(i)] ^= b2i(i);
00148
```

4.1.2.14 void ABitSet::InvertAll ()

Definition at line 221 of file abitset.C.

```
00222 {
00223         uint    n = b2c1(num);
00224         for (uint i=0; i<n; ++i)
00225                 data[i] ^= ~0;
00226
```

4.1.2.15 bool ABitSet::IsAllClear () const

Definition at line 198 of file abitset.C.

Referenced by IsEmpty().

```
00199 {
00200         uint    n = b2c1(num);
```

```
00201         if (n>1)
00202                 for (uint i=0; i<n-1; ++i)
00203                         if (data[i] != 0)
00204                                 return 0;
00205         return (data[n-1] & all_set[num%8]) == '\x00';
00206
```

4.1.2.16 bool ABitSet::IsAllSet () const

Definition at line 187 of file abitset.C.

Referenced by bf_iterator::operator++(), and df_iterator::operator++().

```
00188 {
00189         uint    n = b2c1(num);
00190         if (n>1)
00191                 for (uint i=0; i<n-1; ++i)
00192                         if (data[i] != '\xFF')
00193                                 return 0;
00194         return (data[n-1] & all_set[num%8]) == all_set[num%8];
00195
```

4.1.2.17 bool ABitSet::IsEmpty () const [inline]

Definition at line 62 of file abitset.h.

```
00063                         { return IsAllClear(); }
00064
```

4.1.2.18 bool ABitSet::IsSet (uint *i*) const [inline]

Definition at line 66 of file abitset.h.

Referenced by sdGraph::TravLists(), bf_iterator::operator++(), and df_iterator::operator++().

```
00067                                 { return ($\ast$this)[i]; }
00068
```

4.1.2.19 ABitSet::MAKECOMPARISONS_WITH_COMPARE (ABitSet)

4.1.2.20 int ABitSet::Number () const [inline]

Definition at line 102 of file abitset.h.

```
00103                         { return num; }
00104
```

4.1.2.21 void ABitSet::Remove (uint *i*) [inline]

Definition at line 77 of file abitset.h.

```
00078                                 { Clear(i); }
00079
```

4.1.2.22 bool ABitSet::Resize (uint *n*)

Definition at line 80 of file abitset.C.

```
00081 {
00082         if (n<1)
00083                 n = 1;
00084         if (n==num)
00085                 return 1;
00086
00087         int     m = util_min(n,num);
00088         char*   newd = new char[n];
00089         if (!newd)
00090                 return 0;
00091
00092         memcpy (newd, data, m);
00093         delete[] data;
00094         data = newd;
00095         return 1;
00096
```

4.1.2.23 void ABitSet::Set (uint *i*, uint *j*)

Definition at line 151 of file abitset.C.

```
00152 {
00153         if (i>j)
00154                 util_swap(i,j);
00155         if ( j>=num )
00156                 j=num-1;
00157
00158         for (uint k=i; k<=j; ++k)
00159                 Set(k);
00160
```

4.1.2.24 void ABitSet::Set (uint *i*)

Definition at line 124 of file abitset.C.

Referenced by Add(), KAdjMat::Connect(), Set(), bf_iterator::operator++(), df_iterator::operator++(), bf_iterator::reset(), and df_iterator::reset().

```
00125 {
00126         if ( i>=num )
00127                 return;
00128
00129         data[b2c0(i)] |= b2i(i);
00130
```

4.1.2.25 void ABitSet::SetAll ()

Definition at line 209 of file abitset.C.

```
00210 {
00211         memset(data, ~0, b2c1(num));
00212
```

4.1.2.26 ABitSet ABitSet::operator & (const ABitSet & *abs*) const

Definition at line 289 of file abitset.C.

```
00290 {
00291         ABitSet res(*this);
00292         return res &= abs;
00293
```

4.1.2.27 const ABitSet & ABitSet::operator &= (const ABitSet & *abs*)

Definition at line 250 of file abitset.C.

```
00251 {
00252         uint    n = b2c1(util_min(num, abs.num));
00253
00254         for (uint i=0; i<n; ++i)
00255                 data[i] &= abs.data[i];
00256
00257         return *this;
00258
```

4.1.2.28 ABitSet ABitSet::operator- (const ABitSet & *abs*) const

Definition at line 296 of file abitset.C.

```
00297 {
00298         ABitSet res(*this);
00299         return res -= abs;
00300
```

4.1.2.29 const ABitSet & ABitSet::operator-= (const ABitSet & *abs*)

Definition at line 261 of file abitset.C.

```
00262 {
00263         uint    n = b2c1(util_min(num, abs.num));
00264         for (uint i=0; i<n; ++i)
00265                 data[i] &= ~(abs.data[i]);
00266
00267         return *this;
00268
```

4.1.2.30 const ABitSet & ABitSet::operator= (const ABitSet & *abs*)

Definition at line 60 of file abitset.C.

```
00061 {
00062         if (this == &abs)
00063                 return *this;
00064
00065         const uint x=b2c1(abs.num);
00066
00067         if (b2c1(num) < x)
00068         {
```

```
00069                   delete[] data;
00070                   data = new char[x];
00071           }
00072
```

4.1.2.31 bool ABitSet::operator[] (uint *i*) const

Definition at line 115 of file abitset.C.

```
00116 {
00117           if ( i>=num )
00118                   return 0;
00119
00120           return data[b2c0(i)] & b2i(i);
00121
```

4.1.2.32 ABitSet ABitSet::operator^ (const ABitSet & *abs*) const

Definition at line 303 of file abitset.C.

```
00304 {
00305           ABitSet res(*this);
00306           return res ^= abs;
00307
```

4.1.2.33 const ABitSet & ABitSet::operator^= (const ABitSet & *abs*)

Definition at line 271 of file abitset.C.

```
00272 {
00273           uint    n = b2c1(util_min(num, abs.num));
00274
00275           for (uint i=0; i<n; ++i)
00276                   data[i] ^= abs.data[i];
00277
00278           return *this;
00279
```

4.1.2.34 ABitSet ABitSet::operator| (const ABitSet & *abs*) const

Definition at line 282 of file abitset.C.

```
00283 {
00284           ABitSet res(*this);
00285           return res |= abs;
00286
```

4.1.2.35 const ABitSet & ABitSet::operator|= (const ABitSet & *abs*)

Definition at line 239 of file abitset.C.

```
00240 {
00241         uint    n = b2c1(util_min(num, abs.num));
00242
00243         for (uint i=0; i<n; ++i)
00244                 data[i] |= abs.data[i];
00245
00246         return *this;
00247
```

4.1.3 Friends And Related Function Documentation

4.1.3.1 istream& InputBinary (istream & *is*, ABitSet & *bs*) [friend]

Definition at line 448 of file abitset.C.
Referenced by InputBinary(), and operator>=().

```
00449 {
00450         if (!is)
00451                 return is;
00452
00453         uint    n;
00454         InputBinary(is,n);
00455
00456         delete[] bs.data;
00457         bs.num = n;
00458         bs.data = new char[b2c1(n)];
00459         if (!bs.data)
00460                 raise(SIGQUIT);
00461
00462         is.read(bs.data, b2c1(bs.num));
00463         return is;
00464
```

4.1.3.2 ostream& OutputBinary (ostream & *os*, const ABitSet & *bs*) [friend]

Definition at line 436 of file abitset.C.
Referenced by OutputBinary(), and operator<=().

```
00437 {
00438         if (!os)
00439                 return os;
00440         OutputBinary(os,bs.num);
00441         if (!os)
00442                 return os;
00443         os.write(bs.data, b2c1(bs.num));
00444         return os;
00445
```

4.1.3.3 ostream& operator<< (ostream & *os*, const ABitSet & *abs*) [friend]

Definition at line 367 of file abitset.C.

```
00368 {
00369         os << '<';
00370         for (uint i=0; i<abs.num; ++i)
00371                 os << (abs[i] ? '1' : '0');
```

```
00372         os << '>';
00373         return os;
00374
```

4.1.3.4 ostream & operator<= (ostream & *os*, const ABitSet & *bs*) [friend]

Definition at line 113 of file abitset.h.

```
00114                 { return OutputBinary(os,bs); }
00115
```

4.1.3.5 istream & operator>= (istream & *is*, ABitSet & *bs*) [friend]

Definition at line 115 of file abitset.h.

```
00116                 { return InputBinary(is,bs); }
00117
```

4.1.4 Member Data Documentation

4.1.4.1 char * ABitSet::data [private]

Definition at line 22 of file abitset.h.

4.1.4.2 uint ABitSet::default_size = 256 [static, protected]

Definition at line 9 of file abitset.C.

4.1.4.3 uint ABitSet::num [private]

Definition at line 21 of file abitset.h.

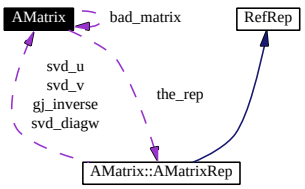
The documentation for this class was generated from the following files:

- **abitset.h**
- **abitset.C**

4.2 AMatrix Class Reference

`#include <amatrix.h>`

Collaboration diagram for AMatrix:



Public Types

- enum **init_type** { **garbage** = -1, **zero** = 0, **ident**, **rand** }
- enum **coord_type** { **rect_c** = 0, **spher_c**, **polar_c** = spher_c, **spher2_c**, **polar2_c** = spher2_c, **cyl_c**, **cyl2_c** }

Public Methods

- **AMatrix** ()
- **AMatrix** (int r, int c, double d)
- **AMatrix** (int r, int c, **init_type** which=garbage)
- **AMatrix** (const AMatrix &m)
- **~AMatrix** ()
- const AMatrix& **operator=** (const AMatrix &)
- AMatrix **UniqueCopy** () const
- AMatrix **Inverse** () const
- AMatrix **SVD_U** () const
- AMatrix **SVD_W** () const
- AMatrix **SVD_DiagW** () const
- AMatrix **SVD_V** () const
- AMatrix **SVDInverse** () const
- AMatrix **SVD_Fit** (int num_funcs, void(*funcer)(double x, double *dest, int nf), double &chisq) const
- AMatrix **GJSolve** (const AMatrix &b) const
- AMatrix **GJInverse** () const
- AMatrix **LinearSolve** (const AMatrix &b) const
- AMatrix **Transpose** () const
- double& **operator()** (int r, int c)
- double **operator()** (int r, int c) const
- double& **operator()** (int r)
- double **operator()** (int r) const
- double& **operator**[ ] (int c)
- double **operator**[ ] (int c) const
- int **Rows** () const
- int **Cols** () const
- int **Compare** (const AMatrix &m) const
- bool **operator==** (const AMatrix &m) const

- bool **operator!=** (const AMatrix &m) const
- bool **operator<** (const AMatrix &m) const
- bool **operator>** (const AMatrix &m) const
- bool **operator<=** (const AMatrix &m) const
- bool **operator>=** (const AMatrix &m) const
- bool **IsBadMatrix** () const
- double **Lp_Distance** (AMatrix m, double p=0.0) const
- double **Lp_Metric** (double p) const
- double **Lp_Distance** (AMatrix m, double p, int minx, int maxx, int miny, int maxy) const
- double **Lp_Metric** (double p, int minx, int maxx, int miny, int maxy) const
- double **ModLp_Distance** (AMatrix m, AMatrix modulus, double p=0.0) const
- double **ModLp_Metric** (AMatrix modulus, double p) const
- double **ModLp_Distance** (AMatrix m, AMatrix modulus, double p, int minx, int maxx, int miny, int maxy) const
- double **ModLp_Metric** (AMatrix modulus, double p, int minx, int maxx, int miny, int maxy) const
- const AMatrix& **operator+=** (double d)
- const AMatrix& **operator-=** (double d)
- const AMatrix& **operator** *= (double d)
- const AMatrix& **operator/=** (double d)
- AMatrix **operator+** (double d) const
- AMatrix **operator-** (double d) const
- AMatrix **operator** * (double d) const
- AMatrix **operator/** (double d) const
- const AMatrix& **operator+=** (const AMatrix &m)
- const AMatrix& **operator-=** (const AMatrix &m)
- const AMatrix& **operator** *= (const AMatrix &m)
- const AMatrix& **operator/=** (const AMatrix &m)
- AMatrix **operator+** (const AMatrix &m) const
- AMatrix **operator-** (const AMatrix &m) const
- AMatrix **operator** * (const AMatrix &m) const
- AMatrix **operator/** (const AMatrix &m) const
- AMatrix **Filter** (double d=**epsilon**, double repl=0.0) const
- AMatrix **ModNormalize** (AMatrix modulus) const
- void **Reset** () const
- AMatrix **Embed** (const AMatrix &m, int r=1, int c=1)
- AMatrix **Submatrix** (int r, int c, int rlen, int clen) const
- AMatrix **ConvertFromRect** (**coord_type** to) const
- AMatrix **ConvertToRect** (**coord_type** from) const
- AMatrix **Convert** (**coord_type** from, **coord_type** to) const

Static Public Methods

- AMatrix **RotationMatrix** (int n, int a, int b, double rad)
- void **Set_Begin_Row** (const char *)
- void **Set_End_Row** (const char *)

Static Public Attributes

- double **epsilon**
- char* **os_begin_row**
- char* **os_end_row**

Protected Methods

- void **calculate_gj_inverse** () const
- void **calculate_svd** () const
- void **mod** ()

Static Protected Methods

- AMatrix **SmallProduct** (const AMatrix &m1, const AMatrix &m2)
- AMatrix **BigProduct** (const AMatrix &m1, const AMatrix &m2)
- void **converttorect** (const AMatrix &, AMatrix &, int, **coord_type**)
- void **convertfromrect** (const AMatrix &, AMatrix &, int, **coord_type**)

Static Protected Attributes

- const int **product_threshold**
- const AMatrix **bad_matrix**
- double **bad_val**

Private Attributes

- **AMatrixRep*** **the_rep**

Friends

- AMatrix **operator+** (double d, const AMatrix &m)
- AMatrix **operator-** (double d, const AMatrix &m)
- AMatrix **operator** * (double d, const AMatrix &m)
- AMatrix **operator/** (double d, const AMatrix &m)
- ostream& **operator<<** (ostream &os, const AMatrix &m)
- istream& **operator>>** (istream &is, AMatrix &m)
- ostream& **operator<=** (ostream &, const AMatrix &)
- istream& **operator>=** (istream &, AMatrix &)
- ostream& **OutputBinary** (ostream &, const AMatrix &)
- istream& **InputBinary** (istream &, AMatrix &)

4.2.1 Member Enumeration Documentation

4.2.1.1 enum AMatrix::coord_type

Enumeration values:

rect_c

spher_c

polar_c

spher2_c

polar2_c

cyl_c

cyl2_c

Definition at line 68 of file amatrix.h.

```
00068                           {
00069                                rect_c=0, spher_c, polar_c=spher_c,
00070
```

4.2.1.2 enum AMatrix::init_type

Enumeration values:

garbage

zero

ident

rand

Definition at line 67 of file amatrix.h.

```
00067                           { garbage=-1, zero=0, ident, rand }
00068
```

4.2.2 Constructor & Destructor Documentation

4.2.2.1 AMatrix::AMatrix () [inline]

Definition at line 56 of file amatrix.inc.

```
00057 {
00058         the_rep = new AMatrixRep;
00059
```

4.2.2.2 AMatrix::AMatrix (int *r*, int *c*, double *d*)

4.2.2.3 AMatrix::AMatrix (int *r*, int *c*, init_type *which* = garbage)

4.2.2.4 AMatrix::AMatrix (const AMatrix & *m*) [inline]

Definition at line 60 of file amatrix.inc.

```
00061 {
00062         the_rep = m.the_rep;
00063         the_rep->Reference();
00064
```

4.2.2.5 AMatrix::~AMatrix () [inline]

Definition at line 66 of file amatrix.inc.

```
00067 {
00068         the_rep->Dereference();
00069         the_rep = 0;
00070
```

4.2.3 Member Function Documentation

4.2.3.1 AMatrix AMatrix::BigProduct (const AMatrix & *m1*, const AMatrix & *m2*) [static, protected]

Referenced by operator ∗().

4.2.3.2 int AMatrix::Cols () const [inline]

Definition at line 303 of file amatrix.h.

Referenced by Compare(), GJInverse(), operator ∗(), operator ∗=(), operator()(), operator/(), operator/=(), and operator[]().

```
00303                               { return the_rep->cols; }
00304
```

4.2.3.3 int AMatrix::Compare (const AMatrix & *m*) const [inline]

Definition at line 190 of file amatrix.inc.

Referenced by operator!=(), operator<(), operator<=(), operator==(), operator>(), and operator>=().

```
00191 {
00192         if (the_rep == m.the_rep)
00193                 return 0;
00194
00195         if (Rows() > m.Rows())
00196                 return 1;
00197         else if (Rows() < m.Rows())
00198                 return -1;
00199
00200         if (Cols() > m.Cols())
00201                 return 1;
00202         else if (Cols() < m.Cols())
00203                 return -1;
00204
00205         for (int i=1; i<=the_rep->rows; ++i)
00206                 for (int j=1; j<=the_rep->cols; ++j)
00207                 {
00208                         if ( the_rep->operator()(i,j) > m.the_rep->operator()(i,j) )
```

```
00209                                   return 1;
00210                           else if ( the_rep->operator()(i,j) < m.the_rep->operator()(i,j) )
00211                                   return -1;
00212                   }
00213
```

4.2.3.4 AMatrix AMatrix::Convert (coord_type *from*, coord_type *to*) const

4.2.3.5 AMatrix AMatrix::ConvertFromRect (coord_type *to*) const

4.2.3.6 AMatrix AMatrix::ConvertToRect (coord_type *from*) const

4.2.3.7 AMatrix AMatrix::Embed (const AMatrix & *m*, int *r* = 1, int *c* = 1)

4.2.3.8 AMatrix AMatrix::Filter (double *d* = epsilon, double *repl* = 0.0) const

4.2.3.9 AMatrix AMatrix::GJInverse () const [inline]

Definition at line 287 of file amatrix.inc.

```
00288 {
00289 #ifndef NOERRORCHECKING_MATRIX
00290         if (Rows() != Cols())
00291     {
00292 #       ifdef   DEBUG_MATRIX
00293         log << "Error: " << Rows() << 'x' << Cols() << " GJInverse" << endl;
00294 #       endif
00295         return bad_matrix;
00296     }
00297
```

4.2.3.10 AMatrix AMatrix::GJSolve (const AMatrix & *b*) const

4.2.3.11 AMatrix AMatrix::Inverse () const

Referenced by operator/(), and operator/=().

4.2.3.12 bool AMatrix::IsBadMatrix () const [inline]

Definition at line 184 of file amatrix.inc.

```
00185 {
00186         return (this->the_rep) == (bad_matrix.the_rep);
00187
```

4.2.3.13 AMatrix AMatrix::LinearSolve (const AMatrix & *b*) const

4.2.3.14 double AMatrix::Lp_Distance (AMatrix *m*, double *p*, int *minx*, int *maxx*, int *miny*, int *maxy*) const

4.2.3.15 double AMatrix::Lp_Distance (AMatrix *m*, double *p* = 0.0) const

4.2.3.16 double AMatrix::Lp_Metric (double *p*, int *minx*, int *maxx*, int *miny*, int *maxy*) const

4.2.3.17 double AMatrix::Lp_Metric (double *p*) const

4.2.3.18 double AMatrix::ModLp_Distance (AMatrix *m*, AMatrix *modulus*, double *p*, int *minx*, int *maxx*, int *miny*, int *maxy*) const

4.2.3.19 double AMatrix::ModLp_Distance (AMatrix *m*, AMatrix *modulus*, double *p* = 0.0) const

4.2.3.20 double AMatrix::ModLp_Metric (AMatrix *modulus*, double *p*, int *minx*, int *maxx*, int *miny*, int *maxy*) const

4.2.3.21 double AMatrix::ModLp_Metric (AMatrix *modulus*, double *p*) const

4.2.3.22 AMatrix AMatrix::ModNormalize (AMatrix *modulus*) const

4.2.3.23 void AMatrix::Reset () const [inline]

Definition at line 361 of file amatrix.inc.

```
00362 {
00363         if (the_rep->have_svd_flag)
00364         {
00365                 the_rep->have_svd_flag = 0;
00366                 delete the_rep->svd_u;
00367                 the_rep->svd_u = 0;
00368                 delete the_rep->svd_diagw;
00369                 the_rep->svd_diagw = 0;
00370                 delete the_rep->svd_v;
00371                 the_rep->svd_v = 0;
00372         }
00373         if (the_rep->have_inverse_flag);
00374         {
00375                 the_rep->have_inverse_flag = 0;
00376                 delete the_rep->gj_inverse;
00377                 the_rep->gj_inverse = 0;
00378         }
00379
```

4.2.3.24 AMatrix AMatrix::RotationMatrix (int *n*, int *a*, int *b*, double *rad*) [static]

4.2.3.25 int AMatrix::Rows () const [inline]

Definition at line 300 of file amatrix.h.

Referenced by Compare(), GJInverse(), operator *(), operator *=(), operator()(), operator/(), operator/=(), and operator[]().

```
00300                           { return the_rep->rows; }
00301
```

4.2.3.26 AMatrix AMatrix::SVDInverse () const

4.2.3.27 AMatrix AMatrix::SVD_DiagW () const [inline]

Definition at line 304 of file amatrix.inc.

```
00305 {
00306         if (!the_rep->have_svd_flag)
00307                 calculate_svd();
00308         return *(the_rep->svd_diagw);
00309
```

4.2.3.28 AMatrix AMatrix::SVD_Fit (int *num_funcs*, void(* *funcer*)(double x, double *dest, int nf), double & *chisq*) const

4.2.3.29 AMatrix AMatrix::SVD_U () const [inline]

Definition at line 311 of file amatrix.inc.

```
00312 {
00313         if (!the_rep->have_svd_flag)
00314                 calculate_svd();
00315         return *(the_rep->svd_u);
00316
```

4.2.3.30 AMatrix AMatrix::SVD_V () const [inline]

Definition at line 318 of file amatrix.inc.

```
00319 {
00320         if (!the_rep->have_svd_flag)
00321                 calculate_svd();
00322         return *(the_rep->svd_v);
00323
```

4.2.3.31 AMatrix AMatrix::SVD_W () const

4.2.3.32 void AMatrix::Set_Begin_Row (const char * *s*) [inline, static]

Definition at line 382 of file amatrix.inc.

```
00383 {
00384         if (strlen(s) > strlen(os_begin_row))
00385         {
00386                 delete[] os_begin_row;
00387                 os_begin_row = new char[strlen(s)+1];
00388         }
00389
```

4.2.3.33 void AMatrix::Set_End_Row (const char * *s*) [inline, static]

Definition at line 393 of file amatrix.inc.

```
00394 {
00395         if (strlen(s) > strlen(os_end_row))
00396         {
00397                 delete[] os_end_row;
00398                 os_end_row = new char[strlen(s)+1];
00399         }
00400
```

4.2.3.34 AMatrix AMatrix::SmallProduct (const AMatrix & *m1*, const AMatrix & *m2*) [static, protected]

Referenced by operator *().

4.2.3.35 AMatrix AMatrix::Submatrix (int *r*, int *c*, int *rlen*, int *clen*) const

4.2.3.36 AMatrix AMatrix::Transpose () const

4.2.3.37 AMatrix AMatrix::UniqueCopy () const [inline]

Definition at line 82 of file amatrix.inc.

```
00083 {
00084         AMatrix m;
00085         m.the_rep->Dereference();
00086         m.the_rep = new AMatrixRep(*the_rep);
00087         return m;
00088
```

4.2.3.38 void AMatrix::calculate_gj_inverse () const [protected]

Referenced by GJInverse().

4.2.3.39 void AMatrix::calculate_svd () const [protected]

Referenced by SVD_DiagW(), SVD_U(), and SVD_V().

4.2.3.40 void AMatrix::convertfromrect (const AMatrix &, AMatrix &, int, coord_type) [static, protected]

4.2.3.41 void AMatrix::converttorect (const AMatrix &, AMatrix &, int, coord_type) [static, protected]

4.2.3.42 void AMatrix::mod () [protected]

Referenced by operator *=(), operator()(), operator/=(), and operator[]().

4.2.3.43 AMatrix AMatrix::operator * (const AMatrix & *m*) const [inline]

Definition at line 248 of file amatrix.inc.

```
00249 {
00250 #ifndef NOERRORCHECKING_MATRIX
00251         if (Cols() != m.Rows())
00252     {
00253 #       ifdef   DEBUG_MATRIX
00254         log << "Error: " << Rows() << 'x' << Cols()
00255             << " * " << m.Rows() << 'x' << m.Cols() << endl;
00256 #       endif
00257         return bad_matrix;
00258     }
00259
```

4.2.3.44 AMatrix AMatrix::operator * (double *d*) const

4.2.3.45 const AMatrix & AMatrix::operator *= (const AMatrix & *m*) [inline]

Definition at line 268 of file amatrix.inc.

```
00269 {
00270 #ifndef NOERRORCHECKING_MATRIX
00271         if (Cols() != m.Rows())
00272     {
00273 #       ifdef   DEBUG_MATRIX
00274         log << "Error: " << Rows() << 'x' << Cols()
00275             << " *= " << m.Rows() << 'x' << m.Cols() << endl;
00276 #       endif
00277         return bad_matrix;
00278     }
00279
```

4.2.3.46 const AMatrix& AMatrix::operator *= (double *d*)

4.2.3.47 bool AMatrix::operator!= (const AMatrix & *m*) const [inline]

Definition at line 222 of file amatrix.inc.

```
00223 {
00224         return Compare(m) != 0;
00225
```

4.2.3.48 double AMatrix::operator() (int *r*) const [inline]

Definition at line 90 of file amatrix.inc.

```
00091 {
00092 #ifndef NOERRORCHECKING_MATRIX
00093         if ( (r < 1) || (r > Rows()) )
00094         {
00095 #       ifdef   DEBUG_MATRIX
00096                 log << "Error: " << Rows() << 'x' << Cols()
00097                         << " ( " << r << " ) const" << endl;
00098 #       endif
```

```
00099                 return bad_val;
00100         }
00101
```

4.2.3.49 double & AMatrix::operator() (int r) [inline]

Definition at line 105 of file amatrix.inc.

```
00106 {
00107 #ifndef NOERRORCHECKING_MATRIX
00108         if ( (r < 1) || (r > Rows()) )
00109         {
00110 #       ifdef   DEBUG_MATRIX
00111                 log << "Error: " << Rows() << 'x' << Cols()
00112                         << " ( " << r << " ) " << endl;
00113 #       endif
00114                 return bad_val;
00115         }
00116
```

4.2.3.50 double AMatrix::operator() (int r, int c) const [inline]

Definition at line 152 of file amatrix.inc.

```
00153 {
00154 #ifndef NOERRORCHECKING_MATRIX
00155         if ( (r < 1) || (c < 1) || (r > Rows()) || (c > Cols()) )
00156         {
00157 #       ifdef   DEBUG_MATRIX
00158                 log << "Error: " << Rows() << 'x' << Cols()
00159                         << " ( " << r << ',' << c << " ) const" << endl;
00160 #       endif
00161                 return bad_val;
00162         }
00163
```

4.2.3.51 double & AMatrix::operator() (int r, int c) [inline]

Definition at line 167 of file amatrix.inc.
Referenced by Compare().

```
00168 {
00169 #ifndef NOERRORCHECKING_MATRIX
00170         if ( (r < 1) || (c < 1) || (r > Rows()) || (c > Cols()) )
00171         {
00172 #       ifdef   DEBUG_MATRIX
00173                 log << "Error: " << Rows() << 'x' << Cols()
00174                         << " ( " << r << ',' << c << " ) " << endl;
00175 #       endif
00176                 return bad_val;
00177         }
00178
```

4.2.3.52 AMatrix AMatrix::operator+ (const AMatrix & *m*) const

4.2.3.53 AMatrix AMatrix::operator+ (double *d*) const

4.2.3.54 const AMatrix& AMatrix::operator+= (const AMatrix & *m*)

4.2.3.55 const AMatrix& AMatrix::operator+= (double *d*)

4.2.3.56 AMatrix AMatrix::operator- (const AMatrix & *m*) const

4.2.3.57 AMatrix AMatrix::operator- (double *d*) const

4.2.3.58 const AMatrix& AMatrix::operator-= (const AMatrix & *m*)

4.2.3.59 const AMatrix& AMatrix::operator-= (double *d*)

4.2.3.60 AMatrix AMatrix::operator/ (const AMatrix & *m*) const [inline]

Definition at line 327 of file amatrix.inc.

```
00328 {
00329 #ifndef NOERRORCHECKING_MATRIX
00330         if (Cols() != m.Rows())
00331     {
00332 #       ifdef   DEBUG_MATRIX
00333         log << "Error: " << Rows() << 'x' << Cols()
00334             << " / " << m.Rows() << 'x' << m.Cols() << endl;
00335 #       endif
00336         return bad_matrix;
00337     }
00338
```

4.2.3.61 AMatrix AMatrix::operator/ (double *d*) const

4.2.3.62 const AMatrix & AMatrix::operator/= (const AMatrix & *m*) [inline]

Definition at line 343 of file amatrix.inc.

```
00344 {
00345 #ifndef NOERRORCHECKING_MATRIX
00346         if (Cols() != m.Rows())
00347     {
00348 #       ifdef   DEBUG_MATRIX
00349         log << "Error: " << Rows() << 'x' << Cols()
00350             << " /= " << m.Rows() << 'x' << m.Cols() << endl;
00351 #       endif
00352         return bad_matrix;
00353     }
00354
```

4.2.3.63 const AMatrix& AMatrix::operator/= (double *d*)

4.2.3.64 bool AMatrix::operator< (const AMatrix & *m*) const [inline]

Definition at line 232 of file amatrix.inc.

```
00233 {
00234         return Compare(m) < 0;
00235
```

4.2.3.65 bool AMatrix::operator<= (const AMatrix & m) const [inline]

Definition at line 227 of file amatrix.inc.

```
00228 {
00229         return Compare(m) <= 0;
00230
```

4.2.3.66 const AMatrix & AMatrix::operator= (const AMatrix & m) [inline]

Definition at line 72 of file amatrix.inc.

```
00073 {
00074         if (the_rep == m.the_rep)
00075                 return *this;
00076         the_rep->Dereference();
00077         the_rep = m.the_rep;
00078         the_rep->Reference();
00079         return *this;
00080
```

4.2.3.67 bool AMatrix::operator== (const AMatrix & m) const [inline]

Definition at line 217 of file amatrix.inc.

```
00218 {
00219         return Compare(m) == 0;
00220
```

4.2.3.68 bool AMatrix::operator> (const AMatrix & m) const [inline]

Definition at line 242 of file amatrix.inc.

```
00243 {
00244         return Compare(m) > 0;
00245
```

4.2.3.69 bool AMatrix::operator>= (const AMatrix & m) const [inline]

Definition at line 237 of file amatrix.inc.

```
00238 {
00239         return Compare(m) >= 0;
00240
```

4.2.3.70 double AMatrix::operator[] (int *c*) const [inline]

Definition at line 121 of file amatrix.inc.

```
00122 {
00123 #ifndef NOERRORCHECKING_MATRIX
00124         if ( (c < 1) || (c > Cols()) )
00125         {
00126 #       ifdef   DEBUG_MATRIX
00127                 log << "Error: " << Rows() << 'x' << Cols()
00128                         << " [ " << c << " ] const" << endl;
00129 #       endif
00130                 return bad_val;
00131         }
00132
```

4.2.3.71 double & AMatrix::operator[] (int *c*) [inline]

Definition at line 136 of file amatrix.inc.

```
00137 {
00138 #ifndef NOERRORCHECKING_MATRIX
00139         if ( (c < 1) || (c > Cols()) )
00140         {
00141 #       ifdef   DEBUG_MATRIX
00142                 log << "Error: " << Rows() << 'x' << Cols()
00143                         << " [ " << c << " ] " << endl;
00144 #       endif
00145                 return bad_val;
00146         }
00147
```

4.2.4 Friends And Related Function Documentation

4.2.4.1 istream& InputBinary (istream &, AMatrix &) [friend]

4.2.4.2 ostream& OutputBinary (ostream &, const AMatrix &) [friend]

4.2.4.3 AMatrix operator * (double *d*, const AMatrix & *m*) [friend]

4.2.4.4 AMatrix operator+ (double *d*, const AMatrix & *m*) [friend]

4.2.4.5 AMatrix operator- (double *d*, const AMatrix & *m*) [friend]

4.2.4.6 AMatrix operator/ (double *d*, const AMatrix & *m*) [friend]

4.2.4.7 ostream& operator<< (ostream & *os*, const AMatrix & *m*) [friend]

4.2.4.8 ostream& operator<= (ostream & *os*, const AMatrix & *m*) [friend]

Definition at line 404 of file amatrix.inc.

```
00405 { return OutputBinary(os,m); }
00406
```

4.2.4.9 istream& operator>= (istream & *is*, AMatrix & *m*) [friend]

Definition at line 408 of file amatrix.inc.

```
00409 { return InputBinary(is,m); }
00410
```

4.2.4.10 istream& operator>> (istream & *is*, AMatrix & *m*) [friend]

4.2.5 Member Data Documentation

4.2.5.1 const AMatrix AMatrix::bad_matrix [static, protected]

Definition at line 110 of file amatrix.h.

4.2.5.2 double AMatrix::bad_val [static, protected]

Definition at line 111 of file amatrix.h.

4.2.5.3 double AMatrix::epsilon [static]

Definition at line 128 of file amatrix.h.

4.2.5.4 char * AMatrix::os_begin_row [static]

Definition at line 494 of file amatrix.h.

4.2.5.5 char * AMatrix::os_end_row [static]

Definition at line 495 of file amatrix.h.

4.2.5.6 const int AMatrix::product_threshold [static, protected]

Definition at line 108 of file amatrix.h.

4.2.5.7 AMatrixRep * AMatrix::the_rep [private]

Definition at line 102 of file amatrix.h.

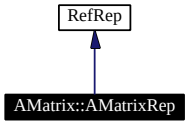
The documentation for this class was generated from the following files:

- **amatrix.h**
- **amatrix.inc**

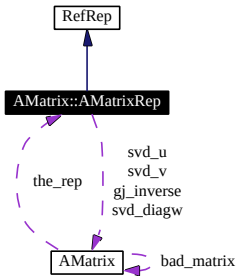
4.3 AMatrix::AMatrixRep Class Reference

`#include <amatrix.h>`

Inheritance diagram for AMatrix::AMatrixRep



Collaboration diagram for AMatrix::AMatrixRep:



Public Methods

- **AMatrixRep** ()
- **AMatrixRep** (const AMatrixRep &mr)
- **AMatrixRep** (int r, int c)
- double& **operator()** (int r, int c)

Protected Methods

- **~AMatrixRep** ()

Static Protected Methods

- double** **do_new** (int r, int c)
- void **do_delete** (double **d, int r, int c)

Private Attributes

- int **rows**
- int **cols**
- double** **d**
- bool **have_svd_flag**
- **AMatrix*** **svd_u**
- **AMatrix*** **svd_diagw**
- **AMatrix*** **svd_v**
- bool **have_inverse_flag**
- **AMatrix*** **gj_inverse**

Friends

- class **AMatrix**

4.3.1 Constructor & Destructor Documentation

4.3.1.1 AMatrix::AMatrixRep::~AMatrixRep () [protected]

4.3.1.2 AMatrix::AMatrixRep::AMatrixRep ()

4.3.1.3 AMatrix::AMatrixRep::AMatrixRep (const AMatrixRep & *mr*)

4.3.1.4 AMatrix::AMatrixRep::AMatrixRep (int *r*, int *c*)

4.3.2 Member Function Documentation

4.3.2.1 void AMatrix::AMatrixRep::do_delete (double ** *d*, int *r*, int *c*) [inline, static, protected]

Definition at line 35 of file amatrix.inc.

```
00036 {
00037         for (int i=1; i<=r; ++i)
00038                 delete[] d[i];
00039         delete[] d;
00040
```

4.3.2.2 double ** AMatrix::AMatrixRep::do_new (int *r*, int *c*) [inline, static, protected]

Definition at line 19 of file amatrix.inc.

```
00020 {
00021         double**        d;
00022         d = new double*[r+1];
00023         if (!d)
00024                 return 0;
00025         for (int i=1; i<=r; ++i)
00026         {
00027                 d[i] = new double[c+1];
00028                 if (!d[i])
00029                         return 0;
00030         }
00031
```

4.3.2.3 double & AMatrix::AMatrixRep::operator() (int *r*, int *c*) [inline]

Definition at line 43 of file amatrix.inc.

```
00044 {
00045         return d[r][c];
00046
```

4.3.3 Friends And Related Function Documentation

4.3.3.1 class AMatrix [friend]

Definition at line 98 of file amatrix.h.

4.3.4 Member Data Documentation

4.3.4.1 int AMatrix::AMatrixRep::cols [private]

Definition at line 76 of file amatrix.h.

4.3.4.2 double ** AMatrix::AMatrixRep::d [private]

Definition at line 77 of file amatrix.h.

4.3.4.3 AMatrix * AMatrix::AMatrixRep::gj_inverse [private]

Definition at line 85 of file amatrix.h.

4.3.4.4 bool AMatrix::AMatrixRep::have_inverse_flag [private]

Definition at line 84 of file amatrix.h.

4.3.4.5 bool AMatrix::AMatrixRep::have_svd_flag [private]

Definition at line 79 of file amatrix.h.

4.3.4.6 int AMatrix::AMatrixRep::rows [private]

Definition at line 76 of file amatrix.h.

4.3.4.7 AMatrix * AMatrix::AMatrixRep::svd_diagw [private]

Definition at line 81 of file amatrix.h.

4.3.4.8 AMatrix * AMatrix::AMatrixRep::svd_u [private]

Definition at line 80 of file amatrix.h.

4.3.4.9 AMatrix * AMatrix::AMatrixRep::svd_v [private]

Definition at line 82 of file amatrix.h.

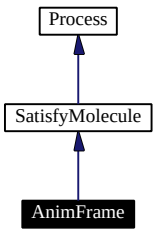
The documentation for this class was generated from the following files:

- **amatrix.h**
- **amatrix.inc**

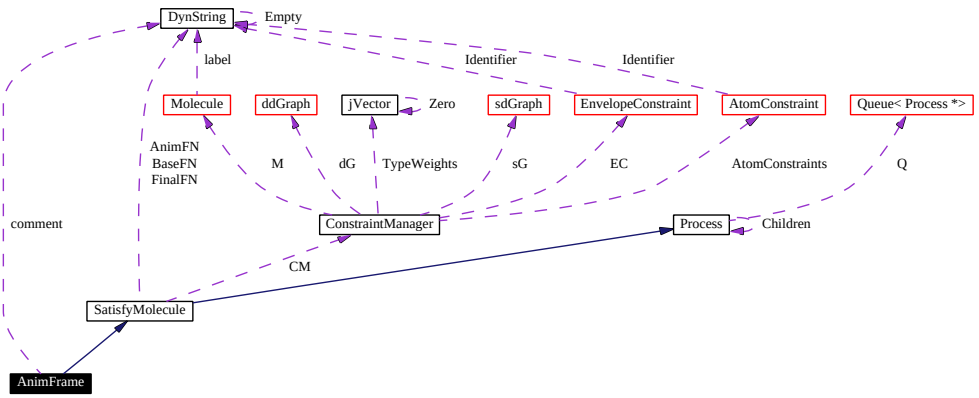
4.4 AnimFrame Class Reference

`#include <jd_anim_frame.h>`

Inheritance diagram for AnimFrame



Collaboration diagram for AnimFrame:



Public Methods

- **AnimFrame** (const AnimFrame &ca)
- **AnimFrame** (**ConstraintManager** &**CM**, **DynString** val="")
- **~AnimFrame** ()
- int **run** ()
- int **run** (ostream *os)
- void **Set_Output_Stream** (**DynString** val)

Public Attributes

- **DynString comment**

Static Public Attributes

- bool **Enabled** = false

Private Attributes

- bool **deleteStream**
- char* **name**
- int* **colors**
- ostream* **Output_Stream**

Static Private Attributes

- int **FrameNum** = 0

4.4.1 Detailed Description

//

File: **jd_anim_frame.h**

Created by: Jon Dugan

Writes a single animation frmae.

//

Definition at line 25 of file jd_anim_frame.h.

4.4.2 Constructor & Destructor Documentation

4.4.2.1 AnimFrame::AnimFrame (const AnimFrame & *ca*)

Definition at line 5 of file jd_anim_frame.C.

```
00006   : SatisfyMolecule(*(ca.CM))
00007 {
00008   // FIX fix copy constructor
00009   DIE("anim frame copy constructor called");
00010
```

4.4.2.2 AnimFrame::AnimFrame (ConstraintManager & *cm_in*, DynString *val* = "")

Definition at line 14 of file jd_anim_frame.C.

```
00015   : SatisfyMolecule(cm_in)
00016 {
00017   //  ostream *Output_Stream = &cout; // default
00018   Set_Output_Stream(val);
00019   int atoms =  cm_in.numAtoms();
00020   colors = new int[atoms];
00021   comment = "--";
00022
```

4.4.2.3 AnimFrame::~AnimFrame ()

Definition at line 26 of file jd_anim_frame.C.

```
00026                     {
00027
00028   delete [] colors;
00029   Output_Stream->flush();
00030
00031   //  if (Output_Stream != &cerr && Output_Stream != &cout)
00032   if (deleteStream)
00033     delete Output_Stream;
00034
00035
```

4.4.3 Member Function Documentation

4.4.3.1 void AnimFrame::Set_Output_Stream (DynString *val*)

Definition at line 37 of file jd_anim_frame.C.
Referenced by AnimFrame().

```
00037                                              {
00038   deleteStream = false;
00039   if (val == DynString("cerr") ||(val.Length() == 0))
00040     Output_Stream = &cerr;
00041   else if (val == DynString("cout"))
00042     Output_Stream = &cout;
00043   else {
00044     if (val[0] == '-'){
00045       cerr << "Confused by filename " << val << " . Try -help option." << endl;
00046       DIE("");
00047     }
00048     deleteStream = true;
00049     char * s = &(val[0]);  // ACK  not good, but it doesn't leak. -jd
00050     Output_Stream = new ofstream(s);
00051   }
00052
```

4.4.3.2 int AnimFrame::run (ostream * *os*)

4.4.3.3 int AnimFrame::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 68 of file jd_anim_frame.C.
Referenced by jdRand_Eval::run(), jdIterator::run(), jdErr_Eval::run(), and AtomBuildup::run().

```
00068                    {
00069 #ifdef JOB_VERBOSE
00070   cerr << "Start: \tAnimFrame::run()" << endl;
00071 #endif
00072
00073   if (AnimFrame::Enabled){
00074     jVector StressVals = CM->getStressList();
00075
00076     for (int i =0; i < StressVals.Length(); i++){
00077       int color =  (int) StressVals[i];
00078       if (color > 9) color = 9;
00079       if (color < 0) color = 0;
00080       colors[i] = color;
```

```
00081     }
00082     AnimFrame::FrameNum++;
00083     CM->getMolecule().writeXmolData(*Output_Stream, colors, comment);
00084   }
00085
```

4.4.4 Member Data Documentation

4.4.4.1 bool AnimFrame::Enabled = false [static]

Definition at line 37 of file globals.C.

4.4.4.2 int AnimFrame::FrameNum = 0 [static, private]

Definition at line 3 of file jd_anim_frame.C.

4.4.4.3 ostream * AnimFrame::Output_Stream [private]

Definition at line 59 of file jd_anim_frame.h.

4.4.4.4 int * AnimFrame::colors [private]

Definition at line 58 of file jd_anim_frame.h.

4.4.4.5 DynString AnimFrame::comment

Definition at line 46 of file jd_anim_frame.h.

4.4.4.6 bool AnimFrame::deleteStream [private]

Definition at line 55 of file jd_anim_frame.h.

4.4.4.7 char * AnimFrame::name [private]

Definition at line 56 of file jd_anim_frame.h.

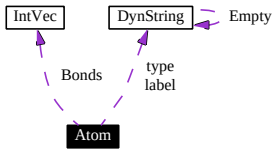
The documentation for this class was generated from the following files:

- **jd_anim_frame.h**
- **globals.C**
- **jd_anim_frame.C**

4.5 Atom Class Reference

`#include <atom.h>`

Collaboration diagram for Atom:



Public Methods

- void **AddBond** (int other_id)
- bool **Bonded** (int other_id) const
- const int& **ID** () const
- int& **ID** ()
- const **DynString**& **Type** () const
- **DynString**& **Type** ()
- const **DynString**& **Label** () const
- **DynString**& **Label** ()
- const double& **VDW** () const
- double& **VDW** ()
- const int& **ResNum** () const
- int& **ResNum** ()
- const double& **cov_dist** () const
- double& **cov_dist** ()
- const bool& **Moved** () const
- bool& **Moved** ()
- const **jVector**& **operator** * () const
- **jVector**& **operator** * ()
- int **Compare** (const Atom &a) const
- const Atom& **operator=** (const Atom &a)
- int **Output** (ostream &os) const
- int **Input** (istream &is)
- int **OutputBinary** (ostream &os) const
- int **InputBinary** (istream &is)
- **MAKECOMPARISONS_WITH_COMPARE** (Atom)
- **MAKE_IO_STREAM_OPERATORS** (Atom)
- **MAKE_BIN_STREAM_OPERATORS** (Atom)

Private Methods

- **Atom** (int identifier=UNINITIALIZED_ATOM_ID)
- **Atom** (const Atom &a)

Private Attributes

- **DynString label**
- **DynString type**
- int **id**
- double **VDW_Radius**
- double **cov_dist_Radius**
- bool **moved**
- int **residue_number**
- **IntVec Bonds**

Friends

- class **Molecule**
- class **Residue**

4.5.1 Constructor & Destructor Documentation

4.5.1.1 Atom::Atom (int *identifier* = UNINITIALIZED_ATOM_ID) [inline, private]

Definition at line 100 of file atom.h.

```
00101 {
00102   id = identifier;
00103   type = "";
00104   label = "uninitialized atom";
00105   moved =  false;
00106   VDW_Radius = 0;
00107   cov_dist_Radius = 0;
00108   residue_number = -1;
00109   Bonds.Resize(0);
00110
00111
```

4.5.1.2 Atom::Atom (const Atom & *a*) [inline, private]

Definition at line 114 of file atom.h.

```
00115 {
00116   id = a.id;
00117   type = a.type;
00118   label = a.label;
00119   VDW_Radius = a.VDW_Radius;
00120   cov_dist_Radius = a.cov_dist_Radius;
00121   moved = a.moved;
00122   residue_number = a.residue_number;
00123   Bonds = a.Bonds;
00124
00125
```

4.5.2 Member Function Documentation

4.5.2.1 void Atom::AddBond (int *other_id*) [inline]

Definition at line 141 of file atom.h.

```
00141                                    {
00142   Bonds.AppendUnique(other_id);
00143
```

4.5.2.2 bool Atom::Bonded (int *other_id*) const [inline]

Definition at line 145 of file atom.h.

```
00145                                             {
00146   return Bonds.Contains(other_id);
00147
```

4.5.2.3 int Atom::Compare (const Atom & *a*) const [inline]

Definition at line 194 of file atom.h.

```
00195 { return (id-a.id); }
00196
```

4.5.2.4 int & Atom::ID () [inline]

Definition at line 151 of file atom.h.

```
00152 { return id; }
00153
```

4.5.2.5 const int & Atom::ID () const [inline]

Definition at line 149 of file atom.h.
Referenced by Molecule::Allocate(), Molecule::Molecule(), and AtomRegistry::SetType().

```
00150 { return id; }
00151
```

4.5.2.6 int Atom::Input (istream & *is*) [inline]

Definition at line 201 of file atom.h.

```
00202 { is >> id >> VDW_Radius >> cov_dist_Radius  >> moved ; return id; }
00203
```

4.5.2.7 int Atom::InputBinary (istream & *is*) [inline]

Definition at line 208 of file atom.h.

```
00209     { is >= id; is >= pos; return id; }
00210
```

4.5.2.8 DynString & Atom::Label () [inline]

Definition at line 156 of file atom.h.

```
00157 { return label; }
00158
```

4.5.2.9 const DynString & Atom::Label () const [inline]

Definition at line 154 of file atom.h.
Referenced by AtomRegistry::SetType().

```
00155 { return label; }
00156
```

4.5.2.10 Atom::MAKECOMPARISONS_WITH_COMPARE (Atom)

4.5.2.11 Atom::MAKE_BIN_STREAM_OPERATORS (Atom)

4.5.2.12 Atom::MAKE_IO_STREAM_OPERATORS (Atom)

4.5.2.13 bool & Atom::Moved () [inline]

Definition at line 181 of file atom.h.

```
00182 { return moved; }
00183
```

4.5.2.14 const bool & Atom::Moved () const [inline]

Definition at line 179 of file atom.h.

```
00180 { return moved; }
00181
```

4.5.2.15 int Atom::Output (ostream & *os*) const [inline]

Definition at line 197 of file atom.h.

```
00198     { os << id << ' ' << pos; return id; }
00199
```

4.5.2.16 int Atom::OutputBinary (ostream & *os*) const [inline]

Definition at line 204 of file atom.h.

```
00205     { os <= id; os <= pos; return id; }
00206
```

4.5.2.17 int & Atom::ResNum () [inline]

Definition at line 171 of file atom.h.

```
00172 {  return residue_number; }
00173
```

4.5.2.18 const int & Atom::ResNum () const [inline]

Definition at line 169 of file atom.h.

Referenced by AtomRegistry::SetType().

```
00170 {  return residue_number; }
00171
```

4.5.2.19 DynString & Atom::Type () [inline]

Definition at line 161 of file atom.h.

```
00162 { return type; }
00163
```

4.5.2.20 const DynString & Atom::Type () const [inline]

Definition at line 159 of file atom.h.

Referenced by AtomRegistry::SetType().

```
00160 { return type; }
00161
```

4.5.2.21 double & Atom::VDW () [inline]

Definition at line 166 of file atom.h.

```
00167 {  return VDW_Radius; }
00168
```

4.5.2.22 const double & Atom::VDW () const [inline]

Definition at line 164 of file atom.h.

Referenced by Envelope::Build(), AtomRegistry::SetType(), and Molecule::VDW().

```
00165 {  return VDW_Radius; }
00166
```

4.5.2.23 double & Atom::cov_dist () [inline]

Definition at line 176 of file atom.h.

```
00177 { return cov_dist_Radius; }
00178
```

4.5.2.24 const double & Atom::cov_dist () const [inline]

Definition at line 174 of file atom.h.
Referenced by AtomRegistry::SetType().

```
00175 { return cov_dist_Radius; }
00176
```

4.5.2.25 jVector& Atom::operator * ()

4.5.2.26 const jVector& Atom::operator * () const

4.5.2.27 const Atom & Atom::operator= (const Atom & *a*) [inline]

Definition at line 127 of file atom.h.

```
00127                                                {
00128
00129   id = a.id;
00130   type = a.type;
00131   label = a.label;
00132   VDW_Radius = a.VDW_Radius;
00133   cov_dist_Radius = a.cov_dist_Radius;
00134   moved = a.moved;
00135   residue_number = a.residue_number;
00136   Bonds = a.Bonds;
00137
00138   return *this;
00139
```

4.5.3 Friends And Related Function Documentation

4.5.3.1 class Molecule [friend]

Definition at line 35 of file atom.h.

4.5.3.2 class Residue [friend]

Definition at line 36 of file atom.h.

4.5.4 Member Data Documentation

4.5.4.1 IntVec Atom::Bonds [private]

Definition at line 96 of file atom.h.

4.5.4.2 double Atom::VDW_Radius [private]

Definition at line 92 of file atom.h.

4.5.4.3 double Atom::cov_dist_Radius [private]

Definition at line 93 of file atom.h.

4.5.4.4 int Atom::id [private]

Definition at line 91 of file atom.h.

4.5.4.5 DynString Atom::label [private]

Definition at line 89 of file atom.h.

4.5.4.6 bool Atom::moved [private]

Definition at line 94 of file atom.h.

4.5.4.7 int Atom::residue_number [private]

Definition at line 95 of file atom.h.

4.5.4.8 DynString Atom::type [private]

Definition at line 90 of file atom.h.

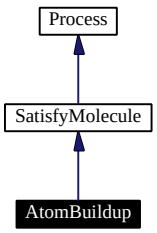
The documentation for this class was generated from the following file:

- **atom.h**

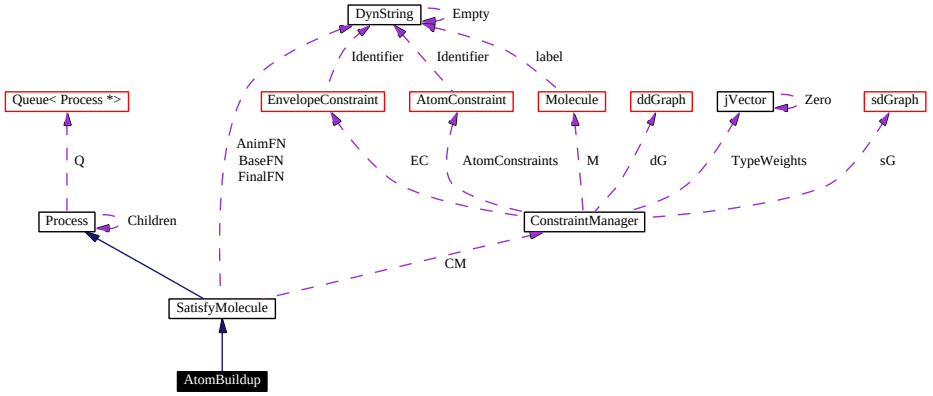
4.6 AtomBuildup Class Reference

`#include <atom_buildup.h>`

Inheritance diagram for AtomBuildup



Collaboration diagram for AtomBuildup:



Public Methods

- **AtomBuildup** (const AtomBuildup &ca)
- **AtomBuildup** (**ConstraintManager** &**CM**)
- **~AtomBuildup** ()
- int **run** ()

Static Public Attributes

- int **MaxIterations** = 50
- int **MaxFinalIterations** = 1000

Private Attributes

- int **IterationNumber**

4.6.1 Detailed Description

//

File: **atom_buildup.h**

Created by: Jon Dugan

AtomBuildup job. iterates over all atoms based on error

//

Definition at line 28 of file atom_buildup.h.

4.6.2 Constructor & Destructor Documentation

4.6.2.1 AtomBuildup::AtomBuildup (const AtomBuildup & *ca*)

Definition at line 3 of file atom_buildup.C.

```
00004   : SatisfyMolecule(*(ca.CM))
00005 {
00006   DIE("AtomBuildup copy constructor called. ");
00007
```

4.6.2.2 AtomBuildup::AtomBuildup (ConstraintManager & *cm*)

Definition at line 10 of file atom_buildup.C.

```
00011   : SatisfyMolecule(cm)
00012 {
00013
00014
```

4.6.2.3 AtomBuildup::~AtomBuildup ()

Definition at line 16 of file atom_buildup.C.

```
00017   {
00018     // Don't delete my children (since Process takes care of it).
00019     // i.e. dont do for(int i=0; i<numChildren; ++i) delete
00020     // Children[i].  or delete[] Children.
00021
00022     // NB in THIS class I am not adding my children to the queue for
00023     // the parent class, Procss'  As a result, the children need to be
00024     // explicitly deleted.
00025
00026     delete Children[0];
00027     delete Children[1];
00028     //    delete Children[2];
00029     delete Children[3];
00030     // Normally a job would NOT do this --
00031     // ~Process would do it, but I am not adding these to the queue
00032
```

4.6.3 Member Function Documentation

4.6.3.1 int AtomBuildup::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 34 of file atom_buildup.C.

```
00034                     {
00035
00036 #ifdef JOB_VERBOSE
00037   cerr << "Start: \tAtom Buildup::run()" << endl;
00038 #endif
00039
00040   //CM->PrintGraph(cout);
00041
00042   int StopFlag = 0;
00043   int iteration = 0;
00044   int curr_atom = 0;
00045
00046   Children = new Process*[4];
00047
00048   Children[0] = new CycleAll(*CM);
00049   Children[1] = new AnimFrame(*CM, AnimFN);
00050   //  Children[2] = new StopEval(*CM, BaseFN);
00051   Children[3] = new RelaxSubset(*CM, .4);
00052
00053   StopEval * SE = new StopEval(*CM);
00054
00055   jVector JVdelta;
00056
00057   JVdelta[0] = 0.5;
00058   JVdelta[1] = 1.0;
00059   JVdelta[2] = 1.5;
00060
00061   jVector JVzero3(3,0);
00062
00063   jVector away(3,200);
00064   for ( int i = 0;i<CM->numAtoms();i++){
00065     CM->setPosition(i, away); // move them out of the way
00066   }
00067   CM->TurnAtomsOff();
00068   CM->TurnEnvelopeOn();
00069
00070   CM->TurnAtomOn(0);
00071   CM->setPosition(0, JVzero3); // put 0 at origin
00072
00073   curr_atom = 1;
00074   CM->TurnAtomOn(curr_atom);
00075   CM->setPosition(curr_atom, CM->getPosition(curr_atom-1)+JVdelta);
00076
00077   while (curr_atom<(CM->numAtoms()-1)) {
00078
00079 #ifdef JOB_VERBOSE
00080     cerr << "Info: \t\t\t\tAtom Buildup added atom " << curr_atom+1 << endl;
00081 #endif
00082
00083     iteration = StopFlag = 0;
00084     while (!StopFlag && iteration < MaxIterations){
00085
00086 #ifdef JOB_VERBOSE
00087     cerr << "Value: \tAtom Buildup iteration =" << iteration << endl;
00088 #endif
00089
00090     // CYCLE and RELAX
00091     StopFlag = Children[0]->run();
00092
00093
00094       //Children[3]->run();
```

```
00095       //Children[3]->run();
00096
00097
00098       // EVALUATE STOP CONDITION
00099 //     if ((StopFlag = (StopFlag && SE->check_errors(1e-7, 1e-6)))){  // FIX TO BE PARAMS
00100 //
00101 // #ifdef JOB_VERBOSE2
00102 //      cerr << "Value: \tAtom Buildup stopping on  SE->check_errors " << iteration << endl;
00103 // #endif
00104 //
00105 //       }
00106       iteration++;
00107
00108     }
00109
00110     curr_atom++;
00111     CM->TurnAtomOn(curr_atom);
00112     CM->setPosition(curr_atom, CM->getPosition(curr_atom-1)+JVdelta); // put 0 at origin
00113
00114     // WRITE ANIMATION FRAME
00115     Children[1]->run();
00116
00117
00118   }
00119
00120   //  do final cycle
00121   //  CM->TurnAtomOn(curr_atom);
00122   //  CM->setPosition(curr_atom,  CM->getPosition(curr_atom-1)+JVdelta); // put 0 at origin
00123
00124   iteration = StopFlag = 0;
00125   while (!StopFlag && iteration < MaxFinalIterations){
00126     Children[0]->run();
00127        //    StopFlag = (SE->check_stress_max_tot(1e-6, 1e-4)) || (Children[2]->run());
00128     StopFlag = SE->check_stress_max_tot(1e-6, 1e-4);
00129     SE->check_low_error(gwNumerics::MaxAbsDistRes, gwNumerics::AveAbsDistRes);
00130
00131     Children[1]->run();
00132     iteration++;
00133     //       cerr << iteration << " ";
00134   }
00135
```

4.6.4 Member Data Documentation

4.6.4.1 int AtomBuildup::IterationNumber [private]

Definition at line 46 of file atom_buildup.h.

4.6.4.2 int AtomBuildup::MaxFinalIterations = 1000 [static]

Definition at line 57 of file globals.C.

4.6.4.3 int AtomBuildup::MaxIterations = 50 [static]

Definition at line 56 of file globals.C.

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

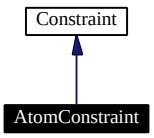
- **atom_buildup.h**

- atom_buildup.C
- globals.C

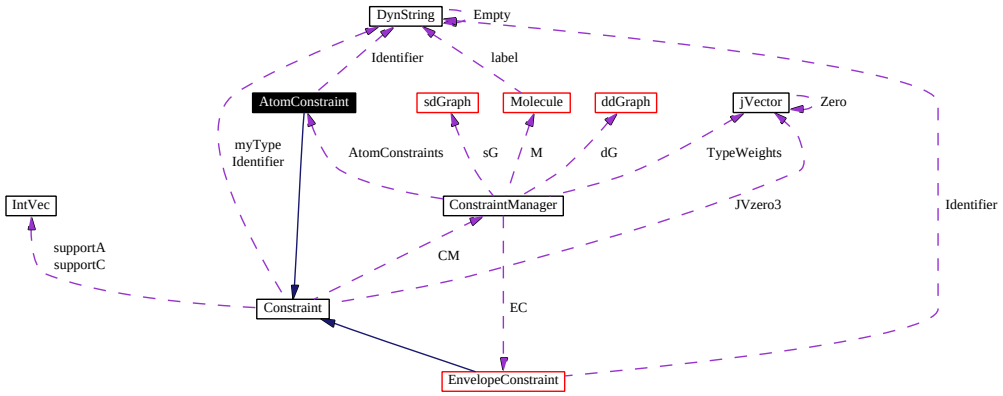
4.7 AtomConstraint Class Reference

`#include <atom_constraint.h>`

Inheritance diagram for AtomConstraint



Collaboration diagram for AtomConstraint:



Public Methods

- **AtomConstraint** (**ConstraintManager** &cm)
- **AtomConstraint** (**ConstraintManager** &cm, int atom_i)
- **~AtomConstraint** ()
- double **computeStress** () const
- double **computeStress** (int k, const **jVector** &pos) const
- **jVector dStress** (int i, const **jVector** &pos_k) const
- int **Compare** (const AtomConstraint &dc) const
- **MAKECOMPARISONS_WITH_COMPARE** (AtomConstraint)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const

Static Public Attributes

- **DynString Identifier** = "atm:"

Protected Methods

- int **Read** (istream &is)
- int **ReadBinary** (istream &is)

  Same scenerio as **Read***()* (p. 57) *only binary version.*

- void **notifyValid** (**Constraint** *child, bool validity=true)
- void **notifyRemove** (void)

  I'm about to be removed from the constraint graph.

4.7.1 Detailed Description

//

atom_constraint.h

This constraint class is really a wrapper class for the atom class so that it can be massaged into a **ConstraintManager** (p. 97)'s constraint graphs.

//

Definition at line 21 of file atom_constraint.h.

4.7.2 Constructor & Destructor Documentation

4.7.2.1 AtomConstraint::AtomConstraint (ConstraintManager & *cm*)

Definition at line 3 of file atom_constraint.C.

```
00004   : Constraint(cm,IntVec(0))
00005   {
00006
00007
```

4.7.2.2 AtomConstraint::AtomConstraint (ConstraintManager & *cm*, int *i*)

Definition at line 9 of file atom_constraint.C.

```
00010   : Constraint(cm, IntVec(0))
00011   {
00012     myID = i;
00013
```

4.7.2.3 AtomConstraint::~AtomConstraint () [inline]

Definition at line 29 of file atom_constraint.h.

```
00029                   {}
00030
```

4.7.3 Member Function Documentation

4.7.3.1 int AtomConstraint::Compare (const AtomConstraint & *ac*) const [inline]

Definition at line 55 of file atom_constraint.h.

```
00056 {
00057
00058   if (myID == ac.ID()) return 0;
00059   if (myID < ac.ID()) return -1;
00060   return 1;
00061
00062   //
00063   //  if(atomIndex[0] == dc.atomIndex[0] && atomIndex[1] == dc.atomIndex[1])
00064   //    return 0;
00065   //  if(atomIndex[0]<dc.atomIndex[0])
00066   //    return -1;
00067   //  if(atomIndex[0]>dc.atomIndex[0])
00068   //    return 1;
00069   //  if(atomIndex[1]<dc.atomIndex[1])
00070   //    return -1;
00071   //  return 1;
00072
```

4.7.3.2 AtomConstraint::MAKECOMPARISONS_WITH_COMPARE (AtomConstraint)

4.7.3.3 int AtomConstraint::Output (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 75 of file atom_constraint.C.

```
00076 {
00077   os << myID;
00078   return 1;
00079
```

4.7.3.4 int AtomConstraint::OutputBinary (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 87 of file atom_constraint.C.

```
00088 {
00089   os <= myID;
00090   return 1;
00091
```

4.7.3.5 int AtomConstraint::Read (istream & *is*) [protected, virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 57) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 57) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented from **Constraint** (p. 85).

Definition at line 81 of file atom_constraint.C.

```
00082 {
```

```
00083   is >> myID;
00084   return 1;
00085
```

4.7.3.6 int AtomConstraint::ReadBinary (istream & *is*) [protected, virtual]

Same scenerio as **Read**() (p. 57) only binary version.

Reimplemented from **Constraint** (p. 85).

Definition at line 106 of file atom_constraint.C.

```
00107 {
00108   is >= myID;
00109   return 1;
00110
```

4.7.3.7 double AtomConstraint::computeStress (int *k*, const jVector & *pos_k*) const [virtual]

Returns the stress on this constraint after "pretending" the atom with index a' moved to position pos_a'. This also must be overwritten by all constraints that inherit from this class

Reimplemented from **Constraint** (p. 89).

Definition at line 42 of file atom_constraint.C.

```
00043 {
00044   if (myID != k)
00045     DIE("Tried to computeStress' on atom constraint for an atom other than the one pointed to by the constraint");
00046   double returnVal = 0;
00047   int numCyc = supportC.Length();
00048   for (int i =0; i < numCyc; i++)
00049     //min variance goes here
00050     returnVal += CM->getConstraint(supportC[i])->stress(myID,pos_k);
00051
00052   return returnVal;
00053
```

4.7.3.8 double AtomConstraint::computeStress () const [virtual]

One of the only functions that *must* be overwritten by subclasses. Returns a value reflecting how dissatisfied this constraint is with the current molecule residing in CM'.

Reimplemented from **Constraint** (p. 89).

Definition at line 20 of file atom_constraint.C.

```
00021 {
00022   // Return the sum of stress of constraints that are "on" me.
00023   // We're not sure we want to do it like this yet.  It violates
00024   // the DAG semantics of constraint use in the constraint graph.
00025   // I.e. Constraints are parents of the constraints whose values they
00026   // use to obtain their stress() value.
00027   //
00028   double returnVal = 0;
00029   int numCyc = supportC.Length();
00030
```

```
00031   for (int i =0; i < numCyc; i++){
00032     returnVal += CM->getConstraint(supportC[i])->stress();
00033 //      cerr << "I'm atom_constraint " << myID << ", Added stress "
00034 //        << CM->getConstraint(supportC[i])->stress() << " from "
00035 //        << supportC[i] <<"("<< CM->getConstraint(supportC[i])->Label()
00036 //        << "), total = " << returnVal << endl;
00037   }
00038
```

4.7.3.9 jVector AtomConstraint::dStress (int *k*, const jVector & *pos_k*) const [virtual]

asking constraint number k for its 12 element dStress vector we pass in pos_a vector – corresponding to the atom position if the constraint is an atom, and corresponding to the position of the atom k in question if the call is to a non-atom constraint. EG for a distance constraint atom 12 needs the dStress from the d-constraint, so it will call dStress on the d-constraint(12, atom_pos) so the d_constr can answer the question if atom 12 were in atom_pos, I would say my dStress is blah.

Reimplemented from **Constraint** (p. 89).

Definition at line 55 of file atom_constraint.C.

```
00056 {
00057   if (myID != k)
00058     DIE("Tried to call dStress on atom constraint for an atom other than the one pointed to by its constraint");
00059   //  jVector ReturnVal = jVector(12,0);
00060   jVector ReturnVal = jVector(3,0);
00061
00062
00063   int numCyc = supportC.Length();
00064
00065   for (int i =0; i < numCyc; i++)
00066     //min variance goes here
00067     if (CM->constraintON(supportC[i])){
00068       //      cerr << "adding to dstress from constraint " << supportC[i] << endl;
00069       ReturnVal += CM->getConstraint(supportC[i])->dStress(myID,pos_k);
00070     }
00071
```

4.7.3.10 void AtomConstraint::notifyRemove (void) [protected, virtual]

I'm about to be removed from the constraint graph.

Reimplemented from **Constraint** (p. 91).

Definition at line 113 of file atom_constraint.C.

```
00114   {
00115     DIE(" Atoms cannot be removed.");
00116
```

4.7.3.11 void AtomConstraint::notifyValid (Constraint * *child*, bool *validity* = true) [protected, virtual]

Informs me that one of my supports now contains valid/invalid information depending on the validity' flag.

Reimplemented from **Constraint** (p. 91).

Definition at line 93 of file atom_constraint.C.

```
00094 {
00095
00096   // this is used to tell the atom that a constraint on which it
00097   // depends (supportC -- up in the graph) has become invalid.
00098
00099   if (validity){
00100     DIE(" Called notifyValid(true) on an Atom constraint.");
00101   }
00102
```

4.7.4 Member Data Documentation

4.7.4.1 DynString AtomConstraint::Identifier = "atm:" [static]

Reimplemented from **Constraint** (p. 95).

Definition at line 94 of file globals.C.

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

- **atom_constraint.h**
- **atom_constraint.C**
- **globals.C**

4.8 AtomRegistry Class Reference

`#include <atom_registry.h>`

Static Public Methods

- int **SetType** (**Atom** &a, const **DynString** &in_type, **DynString** &in_label, int resum=-2)

Static Public Attributes

- const char* **names** [num_known_atom_species]

  names of the atom types.

- const double **covalent_distance** [num_known_atom_species]

  min separation distance to covalently bonded atom.

- const double **radius** [num_known_atom_species]

  VDW radius.

4.8.1 Member Function Documentation

4.8.1.1 int AtomRegistry::SetType (Atom & *a*, const DynString & *in_type*, DynString & *in_label*, int *resnum* = -2) [inline, static]

Definition at line 54 of file atom_registry.h.

Referenced by Residue::AddAtom(), and Molecule::readXmol().

```
00054                                                                                   {
00055   for (int i = 0; i < num_known_atom_species; i++){
00056     if (in_type == names[i]){
00057       if (a.Type() == in_type) {return a.ID();}
00058       if (a.Type().Length() > 0) {cerr <<"|"<< a.Type() <<"|" <<in_type <<"|" << endl; WARN ("atom::SetType failed");
00059       // a.type already set differently
00060       a.Type() = in_type;
00061       a.Label() = in_label;
00062       a.VDW() = radius[i];
00063       a.cov_dist() = covalent_distance[i];
00064       a.ResNum() = resnum;
00065       return a.ID();
00066     }
00067   }
00068 
```

4.8.2 Member Data Documentation

4.8.2.1 const double AtomRegistry::covalent_distance[num_known_atom_species] [static]

Initial value:

```
 {
  0.68, 0.51, 0.47, 0.91, 0.91
}
```

min separation distance to covalently bonded atom.

Definition at line 148 of file globals.C.

4.8.2.2 const char * AtomRegistry::names[num_known_atom_species] [static]

Initial value:

```
 {
  "C","N","O","P","S"
}
```

names of the atom types.

Definition at line 139 of file globals.C.

4.8.2.3 const double AtomRegistry::radius[num_known_atom_species] [static]

Initial value:

```
 {
  1.18,1.1,1,1.25,1.31
}
```

VDW radius.

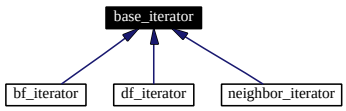
Definition at line 144 of file globals.C.

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

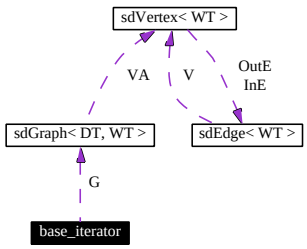
- **atom_registry.h**
- **globals.C**

4.9 base_iterator Class Reference

Inheritance diagram for base_iterator



Collaboration diagram for base_iterator:



Public Methods

- **base_iterator** (int reverse=0)
- **base_iterator** (const base_iterator &bi)
- void **Reverse** ()
- const base_iterator& **operator=** (const base_iterator &bi)
- int **bad** () const
- virtual **~base_iterator** ()
- virtual void **reset** ()
- virtual **operator int** () const
- virtual const base_iterator& **operator++** ()
- virtual int **operator** * () const

Protected Methods

- int **Invalid** (int vndx) const
- **base_iterator** (**sdGraph**< DT, WT > *g, int vndx, int reverse=0)

Protected Attributes

- **sdGraph**<DT,WT>* **G**
- int **vstart**
- int **vid**
- int **rev**

Friends

- class **sdGraph< DT, WT >**

4.9.1 Constructor & Destructor Documentation

4.9.1.1 base_iterator::base_iterator (sdGraph< DT, WT > * *g*, int *vndx*, int *reverse* = 0) [inline, protected]

Definition at line 23 of file sdgiter.inc.

```
00024                 {
00025                   G=g;
00026                   vstart = Invalid(vndx) ? -1 : vndx;
00027                   rev=reverse;
00028                   reset();
00029
```

4.9.1.2 base_iterator::base_iterator (int *reverse* = 0) [inline]

Definition at line 31 of file sdgiter.inc.

```
00032                 { G=0; vstart=-1; vid=-1; rev=reverse; }
00033
```

4.9.1.3 base_iterator::base_iterator (const base_iterator & *bi*) [inline]

Definition at line 33 of file sdgiter.inc.

```
00034                 { G=bi.G; vstart=bi.vstart; vid=bi.vid; rev=bi.rev; }
00035
```

4.9.1.4 base_iterator::~base_iterator () [inline, virtual]

Definition at line 46 of file sdgiter.inc.

```
00046                                       { /*empty*/ }
00047
```

4.9.2 Member Function Documentation

4.9.2.1 int base_iterator::Invalid (int *vndx*) const [inline, protected]

Definition at line 20 of file sdgiter.inc.

Referenced by base_iterator().

```
00021                 { return (!G || (vndx<0) || (vndx>=G->NumV)) ? 1 : 0; }
00022
```

4.9.2.2 void base_iterator::Reverse () [inline]

Definition at line 35 of file sdgiter.inc.

```
00035                               { rev = rev ? 0 : 1; }
00036
```

4.9.2.3 int base_iterator::bad () const [inline]

Definition at line 43 of file sdgiter.inc.

Referenced by path_iterator::operator++(), bf_iterator::reset(), df_iterator::reset(), and neighbor_iterator::reset().

```
00044                 { return (!G || vstart<0); }
00045
```

4.9.2.4 int base_iterator::operator * () const [inline, virtual]

Reimplemented in **neighbor_iterator** (p. 375), **df_iterator** (p. 151), and **bf_iterator** (p. 68).

Definition at line 50 of file sdgiter.inc.

```
00050                                           { return vid; }
00051
```

4.9.2.5 base_iterator::operator int () const [inline, virtual]

Reimplemented in **neighbor_iterator** (p. 375), **df_iterator** (p. 152), and **bf_iterator** (p. 69).

Definition at line 48 of file sdgiter.inc.

```
00048                                          { return (vid!=-1); }
00049
```

4.9.2.6 const base_iterator & base_iterator::operator++ () [inline, virtual]

Reimplemented in **neighbor_iterator** (p. 375), **df_iterator** (p. 152), and **bf_iterator** (p. 69).

Definition at line 49 of file sdgiter.inc.

```
00049                                                  { return *this; }
00050
```

4.9.2.7 const base_iterator & base_iterator::operator= (const base_iterator & *bi*) [inline]

Definition at line 36 of file sdgiter.inc.

```
00037                 {
00038                   if(this!=&bi)
00039                     { G=bi.G; vstart=bi.vstart; vid=bi.vid; rev=bi.rev; }
00040
```

4.9.2.8 void base_iterator::reset () [inline, virtual]

Reimplemented in **neighbor_iterator** (p. 376), **df_iterator** (p. 153), and **bf_iterator** (p. 70).

Definition at line 47 of file sdgiter.inc.

Referenced by base_iterator(), bf_iterator::bf_iterator(), df_iterator::df_iterator(), and neighbor_iterator::neighbor_iterator().

```
00047                                   { vid = vstart; }
00048
```

4.9.3 Friends And Related Function Documentation

4.9.3.1 class sdGraph [friend]

Reimplemented in **neighbor_iterator** (p. 376), **df_iterator** (p. 153), and **bf_iterator** (p. 70).

Definition at line 13 of file sdgiter.inc.

4.9.4 Member Data Documentation

4.9.4.1 sdGraph< DT, WT > * base_iterator::G [protected]

Definition at line 16 of file sdgiter.inc.

4.9.4.2 int base_iterator::rev [protected]

Definition at line 19 of file sdgiter.inc.

4.9.4.3 int base_iterator::vid [protected]

Definition at line 18 of file sdgiter.inc.

4.9.4.4 int base_iterator::vstart [protected]

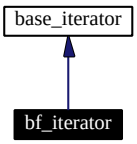
Definition at line 17 of file sdgiter.inc.

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

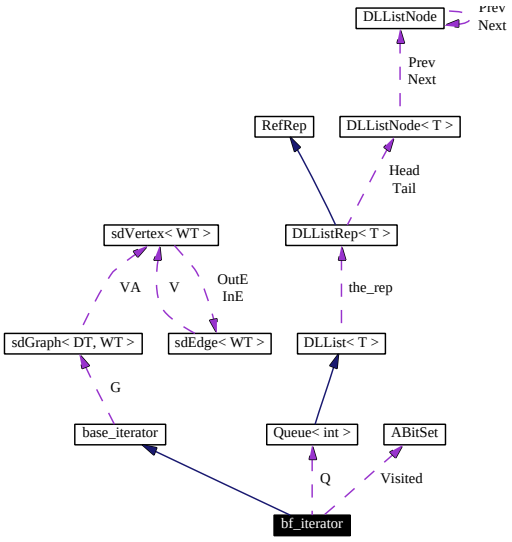
- **sdgiter.inc**

4.10 bf_iterator Class Reference

Inheritance diagram for bf_iterator



Collaboration diagram for bf_iterator:



Public Methods

- **bf_iterator** (int reverse=0)
- **bf_iterator** (const bf_iterator &bfi)
- **~bf_iterator** ()
- const bf_iterator& **operator=** (const bf_iterator &bfi)
- void **reset** ()
- **operator int** () const
- const **base_iterator**& **operator++** ()
- const **base_iterator**& **operator–** ()
- int **operator** * () const

Protected Methods

- **bf_iterator** (**sdGraph**< DT, WT > *g, int vndx, int reverse=0)

Private Attributes

- **ABS* Visited**
- **Queue**<int> **Q**

Friends

- class **sdGraph< DT, WT >**

4.10.1 Constructor & Destructor Documentation

4.10.1.1 bf_iterator::bf_iterator (sdGraph< DT, WT > * *g*, int *vndx*, int *reverse* = 0) [inline, protected]

Definition at line 220 of file sdgiter.inc.

```
00221                           : base_iterator(g,vndx,reverse)
00222
```

4.10.1.2 bf_iterator::bf_iterator (int *reverse* = 0) [inline]

Definition at line 225 of file sdgiter.inc.

```
00225                                       : base_iterator(reverse)
00226
```

4.10.1.3 bf_iterator::bf_iterator (const bf_iterator & *bfi*) [inline]

Definition at line 229 of file sdgiter.inc.

```
00229                                                 : base_iterator(bfi)
00230                   {
00231                     Visited = new ABS(*(bfi.Visited));
00232                     Q = bfi.Q;
00233
```

4.10.1.4 bf_iterator::~bf_iterator () [inline]

Definition at line 235 of file sdgiter.inc.

```
00236                   { delete Visited; }
00237
```

4.10.2 Member Function Documentation

4.10.2.1 int bf_iterator::operator * () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 311 of file sdgiter.inc.

```
00311                                     { return vid; }
00312
```

4.10.2.2 bf_iterator::operator int () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 262 of file sdgiter.inc.

```
00263                   { return (vid!=-1); }
00264
```

4.10.2.3 const base_iterator & bf_iterator::operator++ () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 265 of file sdgiter.inc.

```
00266                   {
00267                     int visited, nbrid;
00268
00269                     // We can stop if we've visited all the vertices:
00270                     if(Visited->IsAllSet()) vid = -1;
00271
00272                     if(vid==-1) return *this;
00273
00274                     const sdEdge<WT>* edges = rev ? G->VA[vid]->InE :
00275                                                     G->VA[vid]->OutE;
00276                     int n = rev ? G->VA[vid]->NumI : G->VA[vid]->NumO;
00277
00278                     // Tack all the unvisited neighbors onto the queue:
00279                     for(int i=0; i<n; ++i)
00280                       {
00281                         nbrid = edges[i].V->ID;
00282                         if(!Visited->IsSet(nbrid))
00283                             Q.Add(nbrid);
00284                       }
00285
00286                     // Skip all the visited vertices in the queue and
00287                     // assign 'vid' to the first unvisited vertex:
00288                     vid = -1;
00289                     visited = 1;
00290                     while(!Q.IsEmpty() && visited)
00291                       {
00292                         vid = Q.Remove();
00293                         if((visited = Visited->IsSet((uint) vid)))
00294                             vid = -1;
00295                       }
00296
```

4.10.2.4 const base_iterator & bf_iterator::operator– () [inline]

Definition at line 303 of file sdgiter.inc.

```
00304                   {
00305                     rev = rev ? 0 : 1;
00306                     ++($\ast$this);
00307                     rev = rev ? 0 : 1;
00308                     return *this;
00309
```

4.10.2.5 const bf_iterator & bf_iterator::operator= (const bf_iterator & *bfi*) [inline]

Definition at line 238 of file sdgiter.inc.

```
00239                   {
00240                     if(this!=&bfi)
00241                       {
00242                         G=bfi.G; vstart=bfi.vstart; vid=bfi.vid; rev=bfi.rev;
00243                         Q = bfi.Q;
00244                         delete Visited;
00245                         Visited = new ABS(*(bfi.Visited));
00246                       }
00247
```

4.10.2.6 void bf_iterator::reset () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).

Definition at line 250 of file sdgiter.inc.

```
00251                   {
00252                     if(bad()) vid=-1;
00253                     else
00254                       {
00255                         Q.Clear();
00256                         vid = vstart;
00257                         Visited->ClearAll();
00258                         Visited->Set((uint) vid);
00259                       }
00260
```

4.10.3 Friends And Related Function Documentation

4.10.3.1 class sdGraph [friend]

Reimplemented from **base_iterator** (p. 66).

Definition at line 217 of file sdgiter.inc.

4.10.4 Member Data Documentation

4.10.4.1 Queue< int > bf_iterator::Q [private]

Definition at line 317 of file sdgiter.inc.

4.10.4.2 ABS * bf_iterator::Visited [private]

Definition at line 314 of file sdgiter.inc.

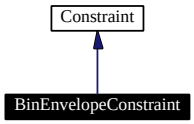
The documentation for this class was generated from the following file:

- **sdgiter.inc**

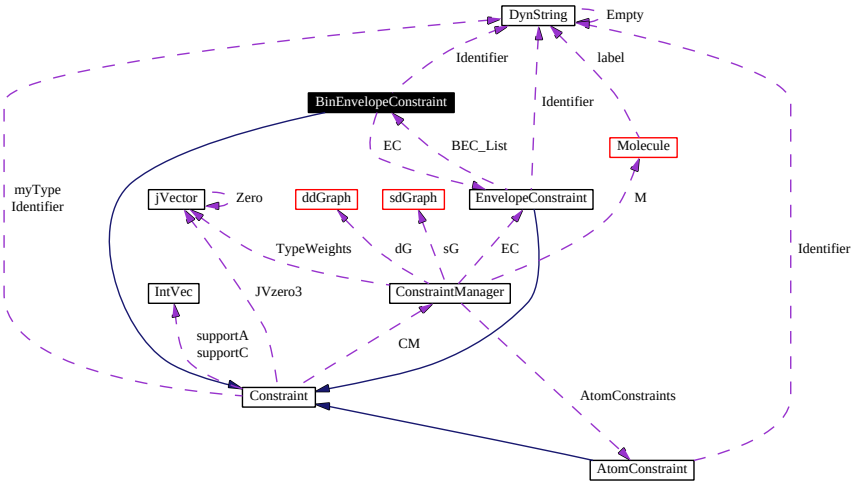
4.11 BinEnvelopeConstraint Class Reference

`#include <bin_envelope_constraint.h>`

Inheritance diagram for BinEnvelopeConstraint



Collaboration diagram for BinEnvelopeConstraint:



Public Methods

- **BinEnvelopeConstraint** (**ConstraintManager** &cm, int atom, **EnvelopeConstraint** *envelope_in)

  Constructor: atom is constraint number for atom, same for envelope.

- **~BinEnvelopeConstraint** ()
- **jVector dStress** (int k, const **jVector** &pos) const

  Returns the (3x1)- actually a 12x1 vector of derivatives of residual with respect to each dimension atom k at position pos.

- int **Compare** (const BinEnvelopeConstraint &ec) const
- **MAKECOMPARISONS_WITH_COMPARE** (BinEnvelopeConstraint)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const

Static Public Attributes

- **DynString Identifier** = "b_env:"

Protected Methods

- double **computeStress** () const

  returns stress on constraint.

- double **computeStress** (int k, const **jVector** &pos) const

  returns stress on constraint if atom k' were in position pos'.

- int **Read** (istream &is)
- int **ReadBinary** (istream &is)

  Same scenerio as **Read***()* (p. 74) *only binary version.*

- void **PreSetup** ()
- void **PostSetup** ()

Protected Attributes

- **EnvelopeConstraint*** **EC**

  passed in – pointing to the parent envelope constraint.

- int **EC_id**
- int **Atom_id**

Friends

- class **EnvelopeConstraint**

4.11.1 Detailed Description

//

bin_envelope_constraint.h

This is a constraint type for GRAPE. It constrains the envelope around a molecule. This is the sub-constraint type which is spawned by **EnvelopeConstraint** (p. 228).

//

Definition at line 23 of file bin_envelope_constraint.h.

4.11.2 Constructor & Destructor Documentation

4.11.2.1 BinEnvelopeConstraint::BinEnvelopeConstraint (ConstraintManager & *cm*, int *atom_in*, EnvelopeConstraint * *envelope_in*)

Constructor: atom is constraint number for atom, same for envelope.

Definition at line 5 of file bin_envelope_constraint.C.

```
00007   : Constraint(cm, IntVec(1,atom_in))  // supported by 1 atom
00008 {
00009
00010   EC = envelope_in;
```

```
00011   EC_id = EC->ID();
00012   Atom_id = atom_in;
00013
00014
```

4.11.2.2 BinEnvelopeConstraint::~BinEnvelopeConstraint ()

Definition at line 16 of file bin_envelope_constraint.C.

```
00016                                           {
00017
00018
```

4.11.3 Member Function Documentation

4.11.3.1 int BinEnvelopeConstraint::Compare (const BinEnvelopeConstraint & *ec*) const [inline]

Definition at line 70 of file bin_envelope_constraint.h.

```
00071 {
00072   DIE("b_env Compare not Implemented");
00073
00074   return 0;
00075
```

4.11.3.2 BinEnvelopeConstraint::MAKECOMPARISONS_WITH_COMPARE (BinEnvelopeConstraint)

4.11.3.3 int BinEnvelopeConstraint::Output (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 124 of file bin_envelope_constraint.C.

```
00125 {
00126   return 1;
00127
00128
```

4.11.3.4 int BinEnvelopeConstraint::OutputBinary (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 136 of file bin_envelope_constraint.C.

```
00137 {
00138   //  os <= supportA <= measured <= variance;
00139   return 1;
00140
```

4.11.3.5 void BinEnvelopeConstraint::PostSetup (void) [protected, virtual]

constraints get their ID when added to the constraint graph. this addition is done in the Input function between the pre- and post setup functions. constraint setup that needs to occur with an ID and an added constraint go in post setup. (like adding parent constraints, etc.)

Reimplemented from **Constraint** (p. 84).

Definition at line 32 of file bin_envelope_constraint.C.

```
00032                                     {
00033
00034
00035
00036
```

4.11.3.6 void BinEnvelopeConstraint::PreSetup (void) [protected, virtual]

The setup function is called if the uninialized constraint method is used to create the constraint to allow different data representations within the constraint. Without this, only data read in from the input file will be available to the member functions. For simple constraints, this is really not very useful. However for multi-atom constraints that need to be a little 'smarter' about data representations, the Setup function will be helpful.

Reimplemented from **Constraint** (p. 85).

Definition at line 21 of file bin_envelope_constraint.C.

```
00021                                     {
00022   myType = BinEnvelopeConstraint::Identifier; // needs to be done here -- not read in by registry
00023   myIntType = ConstraintRegistry::getIntType(myType);
00024
00025   addConstraintSupport(EC_id);
00026   // Valid Envelope constraint is a support
00027   // here valid means atoms are rotated correctly
00028
00029
00030
```

4.11.3.7 int BinEnvelopeConstraint::Read (istream & *is*) [protected, virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 74) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 74) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented from **Constraint** (p. 85).

Definition at line 130 of file bin_envelope_constraint.C.

```
00131 {
00132   // This called on the input stream from envelope_constraint
00133   return 1;
00134
```

4.11.3.8 int BinEnvelopeConstraint::ReadBinary (istream & *is*) [protected, virtual]

Same scenerio as **Read**() (p. 74) only binary version.

Reimplemented from **Constraint** (p. 85).

Definition at line 142 of file bin_envelope_constraint.C.

```
00143 {
00144   // This called on the input stream from envelope_constraint
00145   return 1;
00146
```

4.11.3.9 double BinEnvelopeConstraint::computeStress (int *k*, const jVector & *pos_k*) const [protected, virtual]

returns stress on constraint if atom k' were in position pos'.

Reimplemented from **Constraint** (p. 89).

Definition at line 76 of file bin_envelope_constraint.C.

```
00077 {
00078
00079   return util_max(0.,(MAX_ENV_DEN-(EC->getDensity(pos_k)))/ENV_DEN_RANGE);
00080   DIE("");
00081
```

4.11.3.10 double BinEnvelopeConstraint::computeStress () const [protected, virtual]

returns stress on constraint.

Reimplemented from **Constraint** (p. 89).

Definition at line 40 of file bin_envelope_constraint.C.

```
00040                                         {
00041
00042   double rv = util_max(0.,(MAX_ENV_DEN-(EC->getDensity(getMolecule()(supportA[0])))))/ENV_DEN_RANGE);
00043 //   cerr << "BEC " << myID << ", atom pos = " << getMolecule()(supportA[0]) << endl;
00044 //   cerr << "Calculated BEC stress = " << rv << " atom= " << Atom_id << endl;
00045
00046
00047 //   int atom = supportA[0];
00048 //   jVector pos = getMolecule()(atom);
00049 //   double den = EC->getDensity(pos);
00050 //   double diff = MAX_ENV_DEN-den;
00051 //   double scale = diff/ENV_DEN_RANGE;
00052
00053
00054
00055
00056
00057   return rv;
00058
00059
00063
```

4.11.3.11 jVector BinEnvelopeConstraint::dStress (int *k*, const jVector & *pos_k*) const [virtual]

Returns the (3x1)– actually a 12x1 vector of derivatives of residual with respect to each dimension atom k at position pos.

Reimplemented from **Constraint** (p. 89).

Definition at line 84 of file bin_envelope_constraint.C.

```
00085 {
00086
00087
00088   if(k != supportA[0])
00089     // trying to act on atom other than the one this constraint applies to
00090     {
00091       cerr << "Fatal error: bin_envelope_constraint.C, dStress("<< k \
00092            <<", pos) not legal " << endl;
00093       exit(1);
00094     }
00095
```

4.11.4 Friends And Related Function Documentation

4.11.4.1 class EnvelopeConstraint [friend]

Definition at line 25 of file bin_envelope_constraint.h.

4.11.5 Member Data Documentation

4.11.5.1 int BinEnvelopeConstraint::Atom_id [protected]

Definition at line 50 of file bin_envelope_constraint.h.

4.11.5.2 EnvelopeConstraint * BinEnvelopeConstraint::EC [protected]

passed in – pointing to the parent envelope constraint.

Definition at line 48 of file bin_envelope_constraint.h.

4.11.5.3 int BinEnvelopeConstraint::EC_id [protected]

Definition at line 49 of file bin_envelope_constraint.h.

4.11.5.4 DynString BinEnvelopeConstraint::Identifier = "b_env:" [static]

Reimplemented from **Constraint** (p. 95).

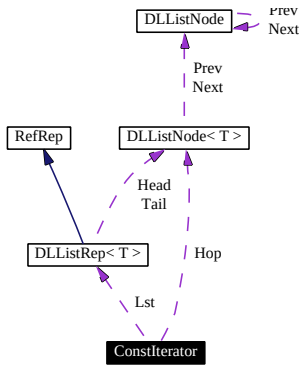
Definition at line 112 of file globals.C.

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

- **bin_envelope_constraint.h**
- **bin_envelope_constraint.C**
- **globals.C**

4.12 ConstIterator Class Reference

Collaboration diagram for ConstIterator:



Public Methods

- void **Reset** ()
- const T& **operator** * ()
- **operator int** () const
- const ConstIterator& **operator++** ()
- void **Copy** (const ConstIterator &it)
- const ConstIterator& **operator=** (const ConstIterator &it)
- **ConstIterator** ()
- **ConstIterator** (const ConstIterator &it)
- **~ConstIterator** ()

Private Methods

- **ConstIterator** (const **DLList**< T > &dll)

Private Attributes

- **DLListRep**<T>* **Lst**
- const **DLListNode**<T>* **Hop**

Friends

- class **DLList**< **T** >

4.12.1 Constructor & Destructor Documentation

4.12.1.1 ConstIterator::ConstIterator () [inline]

Definition at line 69 of file dlliter.inc.

```
00069                               { Lst = 0; Hop = 0; }
00070
```

4.12.1.2 ConstIterator::ConstIterator (const ConstIterator & *it*) [inline]

Definition at line 70 of file dlliter.inc.

```
00071                 { Lst = 0; Hop = 0; Copy(it); }
00072
```

4.12.1.3 ConstIterator::~ConstIterator () [inline]

Definition at line 72 of file dlliter.inc.

```
00073                 { if(Lst) Lst->Dereference(); Lst=0; Hop = 0; }
00074
```

4.12.1.4 ConstIterator::ConstIterator (const DLList< T > & *dll*) [inline, private]

Definition at line 75 of file dlliter.inc.

```
00076                 {
00077                   Lst = dll.the_rep;
00078                   if(Lst) Lst->Reference();
00079                   Reset();
00080
```

4.12.2 Member Function Documentation

4.12.2.1 void ConstIterator::Copy (const ConstIterator & *it*) [inline]

Definition at line 55 of file dlliter.inc.
Referenced by ConstIterator(), and operator=().

```
00056                 {
00057                   if(Lst != it.Lst)
00058                     {
00059                       if(Lst) Lst->Dereference();
00060                       Lst = it.Lst;
00061                       if(Lst) Lst->Reference();
00062                     }
00063
```

4.12.2.2 void ConstIterator::Reset () [inline]

Definition at line 50 of file dlliter.inc.
Referenced by ConstIterator().

```
00050                           { Hop = Lst->Head; }
00051
```

4.12.2.3 const T & ConstIterator::operator * () [inline]

Definition at line 51 of file dlliter.inc.

```
00051                                     { return Hop->Data; }
00052
```

4.12.2.4 ConstIterator::operator int () const [inline]

Definition at line 52 of file dlliter.inc.

```
00052                                    { return Hop!=0; }
00053
```

4.12.2.5 const ConstIterator & ConstIterator::operator++ () [inline]

Definition at line 53 of file dlliter.inc.

```
00054                  { if(Hop) Hop = Hop->Next; return *this; }
00055
```

4.12.2.6 const ConstIterator & ConstIterator::operator= (const ConstIterator & *it*) [inline]

Definition at line 66 of file dlliter.inc.

```
00067                  { Copy(it); return *this; }
00068
```

4.12.3 Friends And Related Function Documentation

4.12.3.1 class DLList [friend]

Definition at line 48 of file dlliter.inc.

4.12.4 Member Data Documentation

4.12.4.1 const DLListNode< T > * ConstIterator::Hop [private]

Definition at line 82 of file dlliter.inc.

4.12.4.2 DLListRep< T > * ConstIterator::Lst [private]

Definition at line 81 of file dlliter.inc.

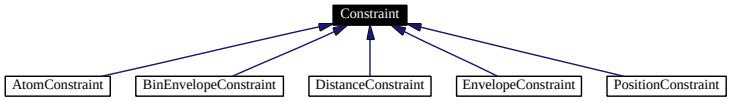
The documentation for this class was generated from the following file:

- **dlliter.inc**

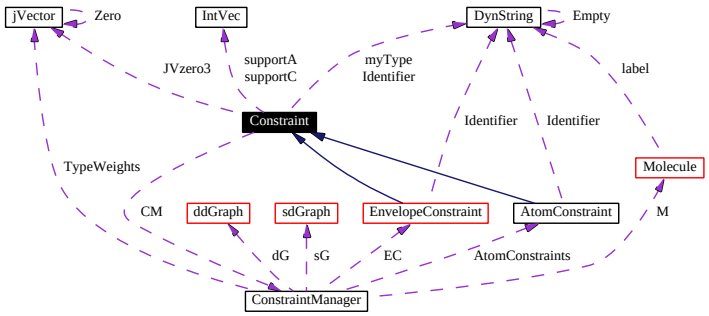
4.13 Constraint Class Reference

`#include <constraint.h>`

Inheritance diagram for Constraint



Collaboration diagram for Constraint:



Public Methods

- virtual **~Constraint** ()
- int **ID** () const

  Returns the id assigned by the **ConstraintManager** (p. 97).

- **DynString Type** () const

  Returns the string type.

- int **IntType** () const

  Returns the string type.

- double **stress** ()

  Returns how dissatisfied this constraint is with the current molecule (stored in the **ConstraintManager** (p. 97)*).*

- double **stress** (int a, const **jVector** &pos_a) const
- virtual **jVector dStress** (int k, const **jVector** &pos_a) const=0
- const **IntVec**& **atomSupport** () const

  Returns the id's of the atoms (wrt the constraint manager) this constraint uses to compute its **stress***()* (p. 92).

- const **IntVec**& **constraintSupport** () const

  Returns the id's of the constraints this constraint uses to compute its **stress***()* (p. 92).

- int **arity** () const

  The number of atoms and/or constraints this constraint uses to compute its **stress***()* (p. 92).

- **MAKECOMPARISONS_NO_COMPARE** (Constraint)

  Give me all the comparison operators.

- virtual int **Output** (ostream &os) const
- virtual int **Input** (istream &is)
- virtual int **OutputBinary** (ostream &os) const
- virtual int **InputBinary** (istream &is)
- **MAKE_IO_STREAM_OPERATORS** (Constraint)
- **MAKE_BIN_STREAM_OPERATORS** (Constraint)
- void **setManager** (**ConstraintManager** &cm)
- const **Molecule**& **getMolecule** () const
- const **ConstraintManager**& **getManager** () const
- bool **isOff** () const

  Returns true if the constraint is off and is being ignored during molecular manipulations.

- virtual void **switchOff** (bool to_off=true)
- bool **isValid** () const
- virtual bool **assertValidity** (bool validity)
- virtual void **addAtomSupport** (int Id)
- virtual void **addConstraintSupport** (int Id)
- virtual double **computeStress** () const=0
- virtual double **computeStress** (int a, const **jVector** &pos_a) const=0
- virtual double **computeAbsRes** () const
- **DynString Label** () const

Static Public Attributes

- **DynString Identifier** = "undefined"

Protected Methods

- **Constraint** (**ConstraintManager** &cm, const **IntVec** &supportAIndices=**IntVec**(0),\const **IntVec** &supportCIndices=**IntVec**(0))
- virtual void **notifyOff** (Constraint *child, bool to_off=true)
- virtual void **notifyValid** (Constraint *child, bool validity)
- virtual void **notifyRemove** ()

  I'm about to be removed from the constraint graph.

- virtual int **Read** (istream &is)=0
- virtual int **ReadBinary** (istream &is)=0

  Same scenerio as **Read***()* (p. 85) *only binary version.*

- virtual void **PreSetup** (void)
- virtual void **PostSetup** (void)

Protected Attributes

- **ConstraintManager* CM**
- **IntVec supportA**
- **IntVec supportC**
- int **myID**
- **DynString myType**

  String representation of this constraint type.

- int **myIntType**

  Int representation of this constraint type.

- double **cached**
- bool **valid**
- bool **off**
- const **jVector JVzero3**

Friends

- class **ConstraintManager**
- class **ConstraintRegistry**

4.13.1 Constructor & Destructor Documentation

4.13.1.1 Constraint::~Constraint () [virtual]

Definition at line 16 of file constraint.C.

```
00017   { CM=0; }
00018
```

4.13.1.2 Constraint::Constraint (ConstraintManager & *cm*, const IntVec & *supportAIndices* = IntVec(0), \const IntVec & *supportCIndices* = IntVec(0)) [protected]

Must be constructed with a **ConstraintManager** (p. 97) (for consistency) and a list of indices specifying which atoms and/or constraints this constraint needs to compute its stress function.

4.13.2 Member Function Documentation

4.13.2.1 int Constraint::ID () const [inline]

Returns the id assigned by the **ConstraintManager** (p. 97).

Definition at line 41 of file constraint.h.

Referenced by BinEnvelopeConstraint::BinEnvelopeConstraint(), AtomConstraint::Compare(), ConstraintManager::TurnEnvelopeOn(), notifyOff(), and switchOff().

```
00041                 { return myID; }
00042
```

4.13.2.2 int Constraint::Input (istream & *is*) [virtual]

Definition at line 446 of file constraint.C.
Referenced by EnvelopeConstraint::PostSetup(), and ConstraintRegistry::readConstraints().

```
00447   {
00448     //    cerr << "Constratint Input Called" << endl;
00449     // Remove myself from the manager:
00450     CM->remove(*this);
00451
00452     // Read my own data from the stream:
00453     int result = Read(is);
00454     // FIX may want to check result here -jd 0 means error result ~= number of reads done.
00455
00456     // Setup any internal stuff the constraint needs to set up before
00457     // being logged with the constraint manager.
00458     PreSetup();
00459
00460     // Add myself back to the manager.  ** ASSIGN an ID **
00461     CM->add(*this);
00462
00463     PostSetup();
00464
00465     // At this point this constraint should be "live" -- ready to go.
00466     // We need to add the existence of this constraint to the
00467     // constraint support of all the atoms in its own atom support.
00468
00469     CM->makeDynamic();
00470     Constraint * c =0;
00471     for (int i = 0; i < supportA.Length(); i++){
00472       c = CM->dG->GetData(supportA[i]);
00474       c->addConstraintSupport(myID);
00475     }
00476
```

4.13.2.3 int Constraint::InputBinary (istream & *is*) [virtual]

Definition at line 481 of file constraint.C.

```
00482   {
00483     // Remove myself from the manager:
00484     CM->remove(*this);
00485
00486     // Read myself in:
00487
00488     int result = ReadBinary(is);
00489
00490     // Setup any internal stuff the constraint wants to
00491     PreSetup();
00492
00493     // Add myself back to the manager:
00494     CM->add(*this);
00495
00496     PostSetup();
00497
00498     return result;
00499
```

4.13.2.4 int Constraint::IntType () const [inline]

Returns the string type.

Definition at line 47 of file constraint.h.

Referenced by ConstraintManager::AbsDistResiduals().

```
00047                       {return myIntType;}
00048
```

4.13.2.5 DynString Constraint::Label () const

Definition at line 416 of file constraint.C.

```
00416                                   {
00417   return myType + " " + myID;
00418
```

4.13.2.6 Constraint::MAKECOMPARISONS_NO_COMPARE (Constraint)

Give me all the comparison operators.

4.13.2.7 Constraint::MAKE_BIN_STREAM_OPERATORS (Constraint)

4.13.2.8 Constraint::MAKE_IO_STREAM_OPERATORS (Constraint)

4.13.2.9 int Constraint::Output (ostream & *os*) const [inline, virtual]

Reimplemented in **AtomConstraint** (p. 57), **BinEnvelopeConstraint** (p. 73), **DistanceConstraint** (p. 156), **EnvelopeConstraint** (p. 234), and **PositionConstraint** (p. 387).

Definition at line 99 of file constraint.h.

```
00100       { os << Constraint::Identifier << supportA << ',' << supportC; return support; }
00101
```

4.13.2.10 int Constraint::OutputBinary (ostream & *os*) const [inline, virtual]

Reimplemented in **AtomConstraint** (p. 57), **BinEnvelopeConstraint** (p. 73), **DistanceConstraint** (p. 156), **EnvelopeConstraint** (p. 235), and **PositionConstraint** (p. 387).

Definition at line 112 of file constraint.h.

```
00113     { return 0; }
00114
```

4.13.2.11 void Constraint::PostSetup (void) [inline, protected, virtual]

constraints get their ID when added to the constraint graph. this addition is done in the Input function between the pre- and post setup functions. constraint setup that needs to occur with an ID and an added constraint go in post setup. (like adding parent constraints, etc.)

Reimplemented in **BinEnvelopeConstraint** (p. 74), and **EnvelopeConstraint** (p. 235).

Definition at line 284 of file constraint.h.

Referenced by Input(), and InputBinary().

```
00284                           {}
00285
```

4.13.2.12 void Constraint::PreSetup (void) [inline, protected, virtual]

The setup function is called if the uninialized constraint method is used to create the constraint to allow different data representations within the constraint. Without this, only data read in from the input file will be available to the member functions. For simple constraints, this is really not very useful. However for multi-atom constraints that need to be a little 'smarter' about data representations, the Setup function will be helpful.

Reimplemented in **BinEnvelopeConstraint** (p. 74), **DistanceConstraint** (p. 156), **EnvelopeConstraint** (p. 235), and **PositionConstraint** (p. 388).

Definition at line 275 of file constraint.h.

Referenced by Input(), and InputBinary().

```
00275                           {}
00276
```

4.13.2.13 int Constraint::Read (istream & *is*) [protected, pure virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 85) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 85) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented in **AtomConstraint** (p. 57), **BinEnvelopeConstraint** (p. 74), **DistanceConstraint** (p. 157), **EnvelopeConstraint** (p. 236), and **PositionConstraint** (p. 388).

Referenced by Input().

4.13.2.14 int Constraint::ReadBinary (istream & *is*) [protected, pure virtual]

Same scenerio as **Read**() (p. 85) only binary version.

Reimplemented in **AtomConstraint** (p. 58), **BinEnvelopeConstraint** (p. 75), **DistanceConstraint** (p. 157), **EnvelopeConstraint** (p. 236), and **PositionConstraint** (p. 388).

Referenced by InputBinary().

4.13.2.15 DynString Constraint::Type () const [inline]

Returns the string type.

Definition at line 44 of file constraint.h.

Referenced by switchOff().

```
00044                          {return myType;}
00045
```

4.13.2.16 void Constraint::addAtomSupport (int *Id*) [virtual]

adds an Id to supportA' ; checks to make sure that the Id is not already in the list.

Definition at line 420 of file constraint.C.

```
00420                                     {
00421   if (myID != UNINITIALIZED_CONSTRAINT_ID) {
00422     // I am already in graph, need to add edge:
00423     CM->dG->AddEdge(this, CM->dG->GetData(Id));
00424
00425     printf ("Constraint::addAtomSupport-> added atom support from constraint %d to %d\n SHOULD THIS OCCUR?", myID, Id
00426
00427   }
00428
```

4.13.2.17 void Constraint::addConstraintSupport (int *Id*) [virtual]

adds an Id to supportC' ; checks to make sure that the Id is not already in the list.

Definition at line 433 of file constraint.C.

Referenced by Input(), and BinEnvelopeConstraint::PreSetup().

```
00433                                          {
00434   if (myID != UNINITIALIZED_CONSTRAINT_ID) {
00435     // I am already in graph, need to add edge:
00436     CM->dG->AddEdge(this, CM->dG->GetData(Id));
00437
00438     //    printf ("Constraint::addConstraintSupport-> added constraint support from constraint %d to %d\n", myID, Id
00439
00440   }
00441
```

4.13.2.18 int Constraint::arity () const [inline]

The number of atoms and/or constraints this constraint uses to compute its **stress**() (p. 92).

Definition at line 93 of file constraint.h.

Referenced by Output().

```
00093                    { return supportA.Length() + supportC.Length(); }
00094
```

4.13.2.19 bool Constraint::assertValidity (bool *validity*) [virtual]

If validity' is true, the constraint is guaranteed to have the correct stress value cached. It will first make sure its supports are valid by calling assertValidity(true) on them (if they're constraints as well). If validity' is false, it notifies its parents that it has been invalidated. The next call to **stress**() (p. 92) will force a recomputation of the cached value. Returns true if the constraint is valid after the call and false if it is invalid. (!!!) This function writes the cached value from the output of computeStress

Definition at line 305 of file constraint.C.

Referenced by ConstraintManager::InvalidateAtoms(), ConstraintManager::InvalidateEnvelope(), notifyOff(), notifyValid(), ConstraintManager::setMolecule(), ConstraintManager::setPosition(), stress(), and switchOff().

```
00306 {
00307   // There is no way to become valid since I'm sleeping.
00308   if(off) {
00309 #ifdef NOISY_CONSTRAINTS
00310     cerr  << "assertValidity:  " << myID <<"(" << myType
00311           <<").  I'm off -- cannot change valid flag " << endl;
00312 #endif
00313
00314     return false;
00315   }
00316     // We agree.
00317   if(valid == validity){
00318
00319 #ifdef NOISY_CONSTRAINTS
00320     cerr  << "assertValidity:  " << myID <<"(" << myType
00321           <<") -- my valid flag is already "<< validity << endl;
00322 #endif
00323
00324     return valid;
00325   }
00326
00327     //
00328     // We disagree. Valid flag will change.
00329     //
00330
00331     // I'm supposed to validate myself; i.e. recompute my stress with valid
00332     // supports.  I need to make sure all the supports are valid first.
00333     //
00334   Constraint * child =0;
00335   if(validity)
00336     {
00337       for(int i=0; i<supportC.Length(); ++i)
00338         {
00339           child = CM->getConstraint(supportC[i]);
00340           if (! child->isOff()){
00341 #ifdef NOISY_CONSTRAINTS
00342             cerr  << "assertValidity:  " << myID <<"(" << myType
00343                   <<"). I'm trying to validate my supports => now I call assertValidity on " << child->myID
00344                   <<"(" << child->myType << ")" << endl;
00345 #endif
00346             if(child->assertValidity(true) == false){
00347 #ifdef NOISY_CONSTRAINTS
00348               cerr << endl << child->myID << " cannot be made valid. " ;
00349 #endif
00350               return false;
00351             } //make child valid
00352           } // if child is on
00353         } // over all child constraints
00354
00355
00356       // Here we're sure that all our supports are valid and we can
00357       // gleefully compute stress().  After this call, I am valid too.
00358       cached = computeStress();
00359       valid = true;
00360     }
00361
00362   // I'm supposed to invalidate myself (perhaps my support has become
00363   // invalid or something).  I need to also notify my parents...
00364   else
```

```
00365     {
00366       valid = false;
00367 //
00368 //      if (CM->dG){
00369 //      cout << "\n\nDYNAMIC VERSION" << endl;
00370 //      for (int i = 0; i <CM->dG->NumVertices();i++){
00371 //        cout << "i=" << i << "\tmyID=" << CM->dG->GetData(i)->ID()<<endl;
00372 //      }
00373 //      cout << endl;
00374 //      }
00375       if (CM->dG){
00376         if(!CM->makeStatic())
00377           {
00378             DIE("Unable to create static graph (constraint.C).\n");
00379           }
00380       }
00381
00382 //      if (CM->sG){
00383 //      cout << "\n\nSTATIC VERSION" << endl;
00384 //      for (int i = 0; i <CM->sG->NumVertices();i++){
00385 //        cout << "i=" << i << "\tmyID=" << CM->sG->GetData(i)->ID()<<endl;
00386 //      }
00387 //      cout << endl;
00388 //      }
00389
00390
00391       // when I am set to false, I must notify the atoms which depend on me
00392       // that I am now invalid
00393       // this is poor, and breaks the assumption about directedness in
00394       // the graph structure.  However, because atoms acually depend
00395       // on constraints up the graph -- this notification is required.
00396       // possibly fix this.
00397       for(int i=0; i<supportA.Length(); ++i)
00398         {
00399           child = CM->getConstraint(supportA[i]);
00400           child->notifyValid(this,false);
00401         }
00402
00403       Constraint * parent=0;
00404       sdGraph<Constraint*>::neighbor_iterator it;
00405       // Iterate over my parents (who I'm a support for):
00406       for(it = CM->sG->begin_neighbor(myID,1); it; ++it)
00407         {
00408           parent = CM->sG->GetData(*it);
00409           parent->notifyValid(this,false);
00410         }
00411     }
00412
```

4.13.2.20 const IntVec & Constraint::atomSupport () const [inline]

Returns the id's of the atoms (wrt the constraint manager) this constraint uses to compute its **stress**() (p. 92).

Definition at line 87 of file constraint.h.

Referenced by ConstraintManager::add().

```
00087                                   { return supportA; }
00088
```

4.13.2.21 double Constraint::computeAbsRes (void) const [inline, virtual]

returns abs residual as a Z score, if supported, constraint overwrites

Reimplemented in **DistanceConstraint** (p. 157).

Definition at line 191 of file constraint.h.

Referenced by ConstraintManager::AbsDistResiduals().

```
00191                                   {return 0;}
00192
```

4.13.2.22 double Constraint::computeStress (int *k*, const jVector & *pos_k*) const [pure virtual]

Returns the stress on this constraint after "pretending" the atom with index a' moved to position pos_a'. This also must be overwritten by all constraints that inherit from this class

Reimplemented in **AtomConstraint** (p. 58), **BinEnvelopeConstraint** (p. 75), **DistanceConstraint** (p. 158), **EnvelopeConstraint** (p. 237), and **PositionConstraint** (p. 388).

4.13.2.23 double Constraint::computeStress () const [pure virtual]

One of the only functions that *must* be overwritten by subclasses. Returns a value reflecting how dissatisfied this constraint is with the current molecule residing in CM'.

Reimplemented in **AtomConstraint** (p. 58), **BinEnvelopeConstraint** (p. 75), **DistanceConstraint** (p. 158), **EnvelopeConstraint** (p. 237), and **PositionConstraint** (p. 389).

Referenced by assertValidity(), and stress().

4.13.2.24 const IntVec & Constraint::constraintSupport () const [inline]

Returns the id's of the constraints this constraint uses to compute its **stress**() (p. 92).

Definition at line 90 of file constraint.h.

Referenced by ConstraintManager::add().

```
00090                                       { return supportC; }
00091
```

4.13.2.25 jVector Constraint::dStress (int *k*, const jVector & *pos_k*) const [pure virtual]

asking constraint number k for its 12 element dStress vector we pass in pos_a vector – corresponding to the atom position if the constraint is an atom, and corresponding to the position of the atom k in question if the call is to a non-atom constraint. EG for a distance constraint atom 12 needs the dStress from the d-constraint, so it will call dStress on the d-constraint(12, atom_pos) so the d_constr can answer the question if atom 12 were in atom_pos, I would say my dStress is blah.

Reimplemented in **AtomConstraint** (p. 59), **BinEnvelopeConstraint** (p. 76), **DistanceConstraint** (p. 159), **EnvelopeConstraint** (p. 237), and **PositionConstraint** (p. 389).

Referenced by ConstraintManager::dStress().

4.13.2.26 const ConstraintManager & Constraint::getManager () const

Definition at line 134 of file constraint.C.

```
00135 { return *CM; }
00136
```

4.13.2.27 const Molecule & Constraint::getMolecule () const

Definition at line 131 of file constraint.C.

Referenced by DistanceConstraint::computeAbsRes(), PositionConstraint::computeStress(), DistanceConstraint::computeStress(), BinEnvelopeConstraint::computeStress(), and DistanceConstraint::dStress().

```
00132 { return CM->getMolecule(); }
00133
```

4.13.2.28 bool Constraint::isOff () const

Returns true if the constraint is off and is being ignored during molecular manipulations.

Definition at line 138 of file constraint.C.

Referenced by assertValidity(), ConstraintManager::constraintON(), and notifyOff().

```
00139   { return off; }
00140
```

4.13.2.29 bool Constraint::isValid () const

isValid() (p. 90) returns true if the cached' value computed from the last **computeStress**() (p. 89) call would equal the value returned by another call to the **computeStress**() (p. 89) function.

Definition at line 302 of file constraint.C.

```
00303   { return valid; }
00304
```

4.13.2.30 void Constraint::notifyOff (Constraint * *child*, bool *to_off* = true) [protected, virtual]

Informs me that one of the atoms/constraints I depend on has been turned off/on depending on the to_off' flag.

Definition at line 253 of file constraint.C.

Referenced by switchOff().

```
00253                                                    {
00254
00255   // default behavior:
00256
00257   if (child->ID() < CM->numAtoms()){
00258     if (to_off){     // a child atom was just turned off
```

```
00259       assertValidity(false);
00260       switchOff(true);
00261       DIE ("This should not be called -- instead, atoms should turn off constraints when turned off");
00262
00263     } else { // a child atom I need was just turned on
00264       if (isOff()) {
00265         // and I am off
00266         // if all my other children constraints (atoms) are on, I turn back on
00267       Constraint * c=0;
00268       sdGraph<Constraint*>::neighbor_iterator it;
00269       for(it = CM->sG->begin_neighbor(myID,0);it; ++it)
00270         {
00271           c = CM->sG->GetData(*it);
00272           if (c->isOff())
00273             return;
00274         }
00275       switchOff(false); // turn back on
00276       } else {
00277         // I am already on
00278         // and one of my child atoms just turned on
00279         DIE("constraint error!");
00280       }
00281     } // end child atom on
00282   }
00283
```

4.13.2.31 void Constraint::notifyRemove (void) [protected, virtual]

I'm about to be removed from the constraint graph.

Reimplemented in **AtomConstraint** (p. 59).

Definition at line 290 of file constraint.C.

Referenced by ConstraintManager::remove().

```
00291 {
00292   Constraint * child;
00293   for(int i=0; i<supportC.Length(); ++i)
00294     {
00295       child = CM->dG->GetData(supportC[i]);
00296       CM->remove(*child);
00297       // Recursively calls notifyRemove() on child''s children.
00298     }
00299
```

4.13.2.32 void Constraint::notifyValid (Constraint * *child*, bool *validity*) [protected, virtual]

Informs me that one of my supports now contains valid/invalid information depending on the validity' flag.

Reimplemented in **AtomConstraint** (p. 59).

Definition at line 239 of file constraint.C.

Referenced by assertValidity(), and ConstraintManager::setPosition().

```
00239                                                       {
00240   // this is DEFAULT behavior.  Many Contraints will overwrite this
00241   // especially complex constraints with many deepndancies.
```

```
00242   // This behavior allows doesnot permit ANY intelligent parsing of
00243   // invalid subsets of children constraints.
00244
00245   // no problem if we match
00246   if (validity == valid) return;
00247
00248   // otherwise fix the problem
00249   if (valid) assertValidity(false);
00250
00251
```

4.13.2.33 void Constraint::setManager (ConstraintManager & *cm*)

Definition at line 113 of file constraint.C.

```
00114   {
00115     if(CM == &cm) return;
00116
00117     // requierd to check CM because most constraints will be
00118     // constructed w/ CM=0 initially and the have their manager set
00119     // with this fntn
00120     if (CM) CM->remove(*this);
00121     CM = &cm;
00122     CM->add(*this);
00123
00124     cached = INF;
00125     valid = false;
00126     off = false;
00127
00128     // My Id has now been set by the ConstraintManager.
00129
```

4.13.2.34 double Constraint::stress (int *a*, const jVector & *pos_a*) const

Computes the stress after simulating atom a' has moved to the new position pos_a'. This function makes sense for values of a' that reference atoms (not other constraints) in the **ConstraintManager** (p. 97).

Definition at line 105 of file constraint.C.

```
00106   {
00107     if(a<0 || a>=CM->numAtoms())
00108         return INF;
00109     if (off) return 0;
00110     return computeStress(a,pos_a); // subclass
00111
```

4.13.2.35 double Constraint::stress ()

Returns how dissatisfied this constraint is with the current molecule (stored in the **ConstraintManager** (p. 97)).

Definition at line 21 of file constraint.C.

Referenced by ConstraintManager::stress().

```
00022   {
00023     if(off)
```

```
00024       return 0;
00025       //       {
00026 // #ifdef NOISY_CONSTRAINTS
00027 //          cerr  << "Constraint::Stress:  " << myID <<"(" << myType
00028 //                <<"). my off value is " << off << endl;
00029 // #endif
00030 //      DIE("stress() called on an inactive constraint.\n");
00031 //       }
00032     //    cerr << "called stress() on constraint" << Label() << endl;
00033
00034     if (assertValidity(true)) return cached;
00035     else {
00036       DIE("Called Constraint::stress() and unable to validate.");
00037       return 0; // removes compiler warning re:control to end of non-void fntn
00038     }
00039
```

4.13.2.36 void Constraint::switchOff (bool *to_off* = true) [virtual]

Turn the constraint on or off. If the constraint is off, it is ignored during subsequent molecular manipulations. Subclasses may want to overwrite these to propigate on/off states to children (constraints they've constructed) when being switched on/off. By default,

Definition at line 141 of file constraint.C.

Referenced by ConstraintManager::TurnAtomOn(), ConstraintManager::TurnAtomsOff(), ConstraintManager::TurnConstraintOn(), and notifyOff().

```
00141                                       {
00142
00143   if(off == to_off) {
00144 #ifdef NOISY_CONSTRAINTS
00145     cerr  << "Turning off value for  " << myID <<"(" << myType
00146           <<").  to " << to_off << "  no change. " << endl;
00147 #endif
00148
00149     return;
00150   }
00151
00152   // I'm turning off
00153   if (to_off) assertValidity(false);
00154
00155 #ifdef NOISY_CONSTRAINTS
00156   cerr  << "I'm  " << myID <<"(" << myType
00157         << ") My off value is now " << to_off << endl;
00158 #endif
00159
00160   off = to_off;
00161
00162   // I just truned on
00163   if (! to_off) assertValidity(false);
00164
00165
00166
00167   CM->makeStatic();
00168   Constraint * c; // Some (child or parent) constraint.
00169   sdGraph<Constraint*>::neighbor_iterator it;
00170
00171
00172   if (myID < CM->numAtoms()){
00173     // I'm an atom
00174
00175     // Switch our child constraints on/off to match our own on/off status.
```

```
00176     for(it = CM->sG->begin_neighbor(myID,0);it; ++it)
00177       {
00178         c = CM->sG->GetData(*it);
00179
00180
00181         if (to_off){ // I'm an atom and just got turned off
00182 #ifdef NOISY_CONSTRAINTS
00183           cerr  << "Atom Constraint  " << myID <<"(" << myType
00184                 <<") now switching  " << c->ID() <<" "  << c->Type()
00185                 << " off value to " << to_off << endl;
00186 #endif
00187           c->switchOff(to_off);
00188
00189         }         else { // I'm an atom and just got turned on
00190 #ifdef NOISY_CONSTRAINTS
00191           cerr  << "Atom Constraint  " << myID <<"(" << myType
00192                 <<") now notifying  " << c->ID() <<" "  << c->Type()
00193                 << " that my off value is now " << to_off << endl;
00194 #endif
00195           c->notifyOff(this, to_off);
00196
00197
00198         }
00199       }
00200
00201   } else {
00202     // I'm a constraint on atoms or on other constraints
00203
00204
00205
00206         // Switch our child constraints on/off to match our own on/off status.
00207     for(it = CM->sG->begin_neighbor(myID,0);it; ++it)
00208       {
00209         c = CM->sG->GetData(*it);
00210
00211         if (c->ID() >= CM->numAtoms()){
00212 #ifdef NOISY_CONSTRAINTS
00213           cerr  << "I'm  " << myID <<"(" << myType
00214                 <<") turning  " << c->ID() <<" "  << c->Type()
00215                 << " off value to " << to_off << endl;
00216 #endif
00217           c->switchOff(to_off);
00218         } else {// else child is an atom, and don't turn off an atom (no loops)
00219           c->assertValidity(false);
00220           // atom must now be invalid b/c a parent turned on
00221         }
00222       }
00223
00224     // Notify the parents that our on/off state has switched.
00225     for(it = CM->sG->begin_neighbor(myID,1);it; ++it)
00226       {
00227         c = CM->sG->GetData(*it);
00228 #ifdef NOISY_CONSTRAINTS
00229         cerr  << "I'm  " << myID <<"(" << myType
00230               <<") notifying  " << c->ID() << " " << c->Type()
00231               << " that my off val is now " << to_off << endl;
00232 #endif
00233         c->notifyOff(this, to_off);
00234       }
00235   }
00236
```

4.13.3 Friends And Related Function Documentation

4.13.3.1 class ConstraintManager [friend]

Definition at line 33 of file constraint.h.

4.13.3.2 class ConstraintRegistry [friend]

Definition at line 34 of file constraint.h.

4.13.4 Member Data Documentation

4.13.4.1 ConstraintManager * Constraint::CM [protected]

Pointer to the manager handling me. If I want to reference a molecule, I must get to it through my constraint manager. This extra abstraction step is meant to guarantee that the manager and I agree on which molecule we're viewing.

Definition at line 204 of file constraint.h.

4.13.4.2 DynString Constraint::Identifier = "undefined" [static]

Reimplemented in **AtomConstraint** (p. 60), **BinEnvelopeConstraint** (p. 76), **DistanceConstraint** (p. 160), **EnvelopeConstraint** (p. 239), and **PositionConstraint** (p. 390).

Definition at line 88 of file globals.C.

4.13.4.3 const jVector Constraint::JVzero3 [protected]

Definition at line 317 of file constraint.h.

4.13.4.4 double Constraint::cached [protected]

Definition at line 299 of file constraint.h.

4.13.4.5 int Constraint::myID [protected]

This id is assigned by the **ConstraintManager** (p. 97) when the constraint is added (during construction).

Definition at line 290 of file constraint.h.

4.13.4.6 int Constraint::myIntType [protected]

Int representation of this constraint type.

Definition at line 296 of file constraint.h.

4.13.4.7 DynString Constraint::myType [protected]

String representation of this constraint type.

Definition at line 293 of file constraint.h.

4.13.4.8 bool Constraint::off [protected]

Definition at line 308 of file constraint.h.

4.13.4.9 IntVec Constraint::supportA [protected]

Stores the id's of the atoms that this constraint needs in order to compute its **stress**() (p. 92) function. It should retrieve these ID's from the constraint manager. The constraint manager assigns integers 0...(N-1) to atoms (where N is the number of atoms in the molecule) and the remainder to constraints.

Definition at line 213 of file constraint.h.

4.13.4.10 IntVec Constraint::supportC [protected]

Stores the id's of the other constraints that this constraint needs in order to compute its **stress**() (p. 92) function. It should retrieve these ID's from the constraint manager. The constraint manager assigns integers 0...(N-1) to atoms (where N is the number of atoms in the molecule) and the remainder to constraints.

Definition at line 222 of file constraint.h.

4.13.4.11 bool Constraint::valid [protected]

Definition at line 304 of file constraint.h.

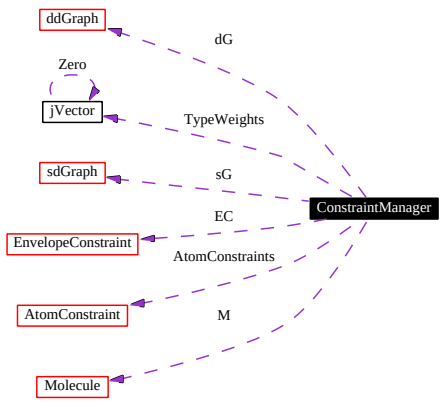
The documentation for this class was generated from the following files:

- **constraint.h**
- **constraint.C**
- **globals.C**

4.14 ConstraintManager Class Reference

`#include <constraint_manager.h>`

Collaboration diagram for ConstraintManager:



Public Methods

- **ConstraintManager** (const **Molecule** &mol)
- virtual ~**ConstraintManager** ()
- int **Number** () const
- int **numAtoms** () const
- int **numAtomsON** () const
- int **numConstraints** () const
- const **Molecule**& **getMolecule** () const
- const **Atom**& **getAtom** (int i) const
- const **jVector**& **getPosition** (int i) const
- **jVector molecule_center** (void) const
- double **getRadius** (int i) const
- **EnvelopeConstraint*** **getEnvelope** () const
- void **InvalidateEnvelope** (void) const
- void **TurnEnvelopeOn** (void) const
- void **InvalidateAtoms** (void) const
- const **Atom**& **operator**[] (int i) const
- const **jVector**& **operator()** (int i) const
- **DLList**<int> **approx_neighbor_i** (const **jVector** &p, double distance) const
- **DLList**<int> **neighbor_i** (const **jVector** &p, double distance) const
- bool **constraintON** (int i) const
- **DLList**<**Constraint***> **getParents** (int Id) const
- **DLList**<**Constraint***> **getAssocAtoms** (int Id) const

  Returns the list of atoms that are involved in some constraint with the atom.

- **Constraint*** **getConstraint** (int id) const

  Returns the pointer to the constraint with id id'.

- ostream& **PrintGraph** (ostream &os)

output a graphical version of the constraint graph for examination.

- virtual double **getWeight** (int id) const
- void **setMolecule** (const **Molecule** &mol)
- void **readNewMoleculeXmol** (ifstream &is)
- void **setPosition** (int i, const **jVector** &v)
- void **setAtom** (int i, const **Atom** &a)
- void **registerEnvelope** (**EnvelopeConstraint** *EC_in)
- void **TurnAtomsOff** () const

  Turn all atoms constratins off.

- void **TurnAtomOn** (int i) const

  Turn single atom on.

- void **TurnConstraintOn** (int i) const
- void **TranslateModel** (**jVector** dir)

  Moves all atoms in model dir.

- void **RandomPerturb** (double dist)

  Perturbs all atoms +/- dist in each direction.

- void **resetMoved** (void)

  set all atoms in molecule to 'not moved yet'.

- virtual void **setWeight** (int id)
- virtual double **CombineStress** (const **IntVec** &IdList, const **jVector** &stressList) const
- double **stress** (int atom)

  Return stress total stress for an atom.

- double **stress** (int atom, const **jVector** &position) const

  Return total stress for an atom as if it were in a new posistion.

- double **stress** ()

  Return sum of stress over all aloms in the molecule.

- **jVector getStressList** ()

  Returns a **jVector** (p. 325) *of the stress values; one for each atom.*

- **jVector dStress** (int atom) const

  Returns the gradient and hessian of stress for an atom.

- **jVector dStress** (int atom, const **jVector** &position) const

  Returns the gradient and hessian of stress for an atom for a new position.

- void **AbsDistResiduals** (double &max, double &ave) const

  Summary Functions.

Protected Types

- typedef **ddGraph<Constraint*> dynamicGraph**
- typedef **sdGraph<Constraint*> staticGraph**

Protected Methods

- bool **isDynamic** () const
- bool **makeStatic** ()
- bool **makeDynamic** ()
- void **remove** (**Constraint** &c)
- int **add** (**Constraint** &c)
- void **clearConstraints** ()

  Erases all the constraints that are not atoms.

- void **clearAllConstraints** ()

  Erases all the constraints, even atoms.

Protected Attributes

- **EnvelopeConstraint* EC**
- **dynamicGraph* dG**

  The dynamic constraint graph.

- **staticGraph* sG**

  The static constraint graph.

- **AtomConstraint* AtomConstraints**

  Array of dummy constraints, each holding one atom in the graph.

- **Molecule M**

  The single molecule this mgr works on.

Private Attributes

- **jVector TypeWeights**

Friends

- class **ConstraintRegistry**
- class **Constraint**

4.14.1 Member Typedef Documentation

4.14.1.1 typedef ddGraph< Constraint *> ConstraintManager::dynamicGraph [protected]

Definition at line 207 of file constraint_manager.h.

Referenced by makeDynamic().

4.14.1.2 typedef sdGraph< Constraint *> ConstraintManager::staticGraph [protected]

Definition at line 208 of file constraint_manager.h.

Referenced by makeStatic().

4.14.2 Constructor & Destructor Documentation

4.14.2.1 ConstraintManager::ConstraintManager (const Molecule & *in_mol*)

Definition at line 4 of file constraint_manager.C.

```
00005 {
00006   M = in_mol;
00007 
00008   //  Weightlist.Resize(M.Number());
00009 
00010   dG = new dynamicGraph;
00011   sG = 0;
00012   EC = 0;
00013 
00014   // assign a constraint to each atom here and add it to the graph.
00015   // assures that constraints numbered 0 -- n-1 are all atom
00016   // constraints
00017   for(int i=0; i<M.Number(); ++i){
00018     add(*(new AtomConstraint(*this)));
00019     getConstraint(i)->myType = AtomConstraint::Identifier;
00020   }
00021 
```

4.14.2.2 ConstraintManager::~ConstraintManager () [virtual]

Definition at line 24 of file constraint_manager.C.

```
00025 {
00026   //
00027   // We *do* remove the constraints since we dont know if we
00028   // were responsible for allocating this space or not (we could have
00029   // read in some constraints from a file, allocated the space for these
00030   // constraints, and then extra constraints could've been added later
00031   // that someone else allocated.  The client should *not* deallocate
00032   // constraints!
00033   //
00034 
00035   makeStatic();
00036 
00037   // cycle through complete graph and delete all constraints
00038   int num = Number();
```

```
00039   for(int i = 0; i < num; i++){
00040     delete (sG->GetData(i));
00041   }
00042
```

4.14.3 Member Function Documentation

4.14.3.1 void ConstraintManager::AbsDistResiduals (double & *max_rv*, double & *ave_rv*) const

Summary Functions.

Definition at line 499 of file constraint_manager.C.

```
00499                                                                    {
00500   int max_c = Number();
00501   int dc_type = ConstraintRegistry::getIntType(DistanceConstraint::Identifier);
00502   int n_dist = 0;
00503
00504   double temp;
00505
00506   max_rv = -1 * INF;
00507   ave_rv = 0;
00508
00509   for (int i = numAtoms();i<max_c;i++){
00510     if (constraintON(i)){
00511       if (getConstraint(i)->IntType() == dc_type){
00512         n_dist++;
00513         temp = getConstraint(i)->computeAbsRes();
00514         //      cerr << getConstraint(i)->Label();
00515         if (max_rv < temp)
00516           max_rv = temp;
00517         ave_rv += temp;
00518       }
00519     }
00520   }
00521
```

4.14.3.2 double ConstraintManager::CombineStress (const IntVec & *IdList*, const jVector & *stressList*) const [inline, virtual]

Definition at line 142 of file constraint_manager.h.

```
00143                                             {
00144     //    not defn yet
00145     return 0;
00146
```

4.14.3.3 void ConstraintManager::InvalidateAtoms (void) const

Definition at line 554 of file constraint_manager.C.

Referenced by readNewMoleculeXmol().

```
00555 {  for (int i = numAtoms()-1;i>=0;i--){
00556   //for (int i =0;i<numAtoms();i++) {
00557   getConstraint(i)->assertValidity(false);}
00558
```

4.14.3.4 void ConstraintManager::InvalidateEnvelope (void) const

Definition at line 528 of file constraint_manager.C.

```
00529 {if (EC) EC->assertValidity(false);}
00530
```

4.14.3.5 int ConstraintManager::Number () const

Definition at line 212 of file constraint_manager.C.

Referenced by AbsDistResiduals(), numConstraints(), ConstraintRegistry::readConstraints(), and ~ConstraintManager().

```
00213 {
00214   if (sG)
00215     return sG->NumVertices();
00216   else
00217     return dG->NumVertices();
00218
```

4.14.3.6 ostream & ConstraintManager::PrintGraph (ostream & *os*)

output a graphical version of the constraint graph for examination.

Definition at line 220 of file constraint_manager.C.

```
00220                                                 {
00221   makeStatic();
00222   sG->PrintDotty(os,0);
00223   return os;
00224
```

4.14.3.7 void ConstraintManager::RandomPerturb (double *dist*)

Perturbs all atoms +/- dist in each direction.

Definition at line 71 of file constraint_manager.C.

```
00072 {
00073
00074   RanDbl randomDist((-1)*dist, dist);
00075   jVector tempPos(3); // 3 element vector
00076
00077     for(int i=0; i<M.Number(); i++)
00078     {
00079       tempPos = getPosition(i);
00080       tempPos[0] += randomDist.Get();
00081       tempPos[1] += randomDist.Get();
00082       tempPos[2] += randomDist.Get();
00083       setPosition(i, tempPos);
00084     }
00085
```

4.14.3.8 void ConstraintManager::TranslateModel (jVector *dir*)

Moves all atoms in model dir.

Definition at line 51 of file constraint_manager.C.

```
00052 {
00053   if (dir.nrm() < 0.0001)
00054     return;
00055
00056   jVector tempV1;
00057   int num = numAtoms();
00058
00059   for (int a = 0; a < num; a++){
00060     if (constraintON(a)){
00061       tempV1 = getPosition(a);
00062       tempV1[0] += dir[0];
00063       tempV1[1] += dir[1];
00064       tempV1[2] += dir[2];
00065       setPosition(a, tempV1);
00066     }
00067   }
00068
```

4.14.3.9 void ConstraintManager::TurnAtomOn (int *i*) const

Turn single atom on.

Definition at line 377 of file constraint_manager.C.

```
00377                                          {
00378   if (i<M.Number()){
00379     cerr  << setw(3)  << i << " ";
00380     if (0 == i%10) {cerr << endl;}
00381
00382 #ifdef NOISY_CONSTRAINTS
00383     cerr  << endl << "CM is turning atom " << i << " on " << endl;
00384 #endif
00385     getConstraint(i)->switchOff(false);
00386   } else {
00387     DIE ("Tried to turn non-atom constraint on");
00388   }
00389
```

4.14.3.10 void ConstraintManager::TurnAtomsOff () const

Turn all atoms constratins off.

Definition at line 365 of file constraint_manager.C.

```
00365                                        {
00366
00367   for(int i=0; i<M.Number(); ++i){
00368 #ifdef NOISY_CONSTRAINTS
00369     cerr  << endl << "CM is turning constaint " << i << " off " << endl;
00370 #endif
00371
00372     getConstraint(i)->switchOff(true);
00373   }
00374
```

4.14.3.11 void ConstraintManager::TurnConstraintOn (int *i*) const

Definition at line 392 of file constraint_manager.C.

Referenced by TurnEnvelopeOn().

```
00392                                                {
00393
00394 #ifdef NOISY_CONSTRAINTS
00395     cerr  << endl << "CM is turning constraint " << i << " on " << endl;
00396 #endif
00397     getConstraint(i)->switchOff(false);
00398
```

4.14.3.12 void ConstraintManager::TurnEnvelopeOn (void) const

Definition at line 531 of file constraint_manager.C.

```
00531                                                 {
00532   if (EC) {
00533     int i = EC->ID();
00534     if (!(constraintON(i))){
00535       TurnConstraintOn(i);
00536     }
00537   }
00538
```

4.14.3.13 int ConstraintManager::add (Constraint & *c*) [protected]

Only friends (e.g. Constraints) and derived classes may add or remove themselves from the manager. The **add**() (p. 104) function returns the id assigned to the newly added constraint.

Definition at line 226 of file constraint_manager.C.

Referenced by ConstraintManager(), Constraint::Input(), Constraint::InputBinary(), and Constraint::setManager().

```
00227 {
00228
00229   makeDynamic();
00230
00231   // Must add all atoms before any constraints on atoms.
00232   // This reserves the indices from 0...(M.Number()-1)
00233   // for use as atom ids.  The rest of the indices are used for
00234   // assigning unique identifiers to constraints.
00235
00236   // Generate an ID for this constraint.
00237   if((c.myID = dG->AddVertex(&c))<0)
00238     {
00239       DIE("ConstraintManager::add-> Problems adding new constraint to constraint graph.\n");
00240       return 1;
00241     }
00242
00243   // Create edges to children' of c'
00244   IntVec children = cat(c.atomSupport(),c.constraintSupport());
00245   Constraint * child;
00246
00247   for(int i=0; i<children.Length(); ++i)
00248     {
00249       // Add an edge from this constraint to the atom/constraint
```

```
00250       // it needs to compute its stress function:
00251
00252       if (children[i]>=0){
00253         // NB uninitialized constraints declare support intvecs with
00254         // appropriate length and set values = -1
00255
00256         child = dG->GetData(children[i]);
00257         dG->AddEdge(&c, child);
00258       }
00259     }
00260
```

4.14.3.14 DLList< int > ConstraintManager::approx_neighbor_i (const jVector & *p*, double *distance*) const [inline]

Definition at line 282 of file constraint_manager.h.

```
00283 {return M.approx_neighbor_i(p,distance); }
00284
```

4.14.3.15 void ConstraintManager::clearAllConstraints () [protected]

Erases all the constraints, even atoms.

Definition at line 424 of file constraint_manager.C.

```
00424                                       {
00425   makeDynamic();
00426     for (int i = (dG->NumVertices())-1; i >=0; i--)
00427       remove(*(dG->GetData(i)));
00428
```

4.14.3.16 void ConstraintManager::clearConstraints () [protected]

Erases all the constraints that are not atoms.

Definition at line 430 of file constraint_manager.C.

Referenced by ConstraintRegistry::readConstraints().

```
00431 {
00432   makeDynamic();
00433     for (int i = (dG->NumVertices())-1; i >=M.Number(); i--)
00434       remove(*(dG->GetData(i)));
00435
```

4.14.3.17 bool ConstraintManager::constraintON (int *i*) const

Definition at line 541 of file constraint_manager.C.

Referenced by AbsDistResiduals(), TranslateModel(), TurnEnvelopeOn(), molecule_center(), numAtomsON(), and setPosition().

```
00541                                           {
00542   //  cerr << "CONSTRAINT " << i << " (off) is " << getConstraint(i)->isOff() << endl;
00543
```

4.14.3.18 jVector ConstraintManager::dStress (int *atom*, const jVector & *position*) const

Returns the gradient and hessian of stress for an atom for a new position.
Definition at line 186 of file constraint_manager.C.

```
00187 {
00188   return getConstraint(atom)->dStress(atom,position) ;
00189
00190 //  /*
00191 //  jVector return_vect(12);
00192 //  // FIX THIS ... THIS IS A TEMPORARY HACK
00193 //  // 0,1,2 are gradient, 3-11 are hessian
00194 //  // The way dStress now works, the CM *has* to also
00195 //  // return the hessian.  Come see me w/ questions -jd
00196 //
00197 //   DLList<Constraint*>::ConstIterator it;
00198 //
00199 //   for(it = AtomConstraintList[atom].GetConstIterator(); it; ++it)
00200 //     return_vect += (*it)->dStress(atom, position);
00201 //   return return_vect;
00202 //  */
00203
```

4.14.3.19 jVector ConstraintManager::dStress (int *atom*) const

Returns the gradient and hessian of stress for an atom.
Definition at line 205 of file constraint_manager.C.

```
00206 {
00207
00208   return getConstraint(atom)->dStress(atom, M(atom));
00209
00210
```

4.14.3.20 DLList< Constraint *> ConstraintManager::getAssocAtoms (int *Id*) const

Returns the list of atoms that are involved in some constraint with the atom.
Definition at line 299 of file constraint_manager.C.

```
00300 {
00301   DLList<Constraint*> ReturnList;
00302
00303   if (Id >= M.Number()) return ReturnList;
00304
00305   DLList<Constraint*> ParentList;
00306   ParentList = getParents(Id);
00307
00308   IntVec atomSeen(M.Number());  // Keeps track of which atoms we've seen.
00309   IntVec atomList;      // List of atom (indices) in a constraint.
00310   int sibling;
00311
00312   atomSeen.zero();      // Mark that we havent seen anything.
00313   atomSeen[Id] = 1; // Mark that we've seen the source atom.
00314
00315   DLList<Constraint *>::ConstIterator it;
```

```
00316
00317   for(it = ParentList.GetConstIterator(); it; ++it)
00318     {
00319       atomList = ($\ast$it)->atomSupport();
00320       for(int i=0; i<atomList.Length(); i++)      {
00321         sibling = atomList[i];  // A mutually-constrained atom index.
00322
00323         // If we havent seen this atom, add it to our list.
00324         if(!atomSeen[sibling])  {
00325           ReturnList.AddToBack(sG->GetData(sibling));
00326           atomSeen[sibling] = 1;
00327         }
00328       }
00329     }
00330
```

4.14.3.21 const Atom & ConstraintManager::getAtom (int *i*) const [inline]

Definition at line 277 of file constraint_manager.h.

Referenced by operator[]().

```
00277                                                   { return M[i]; }
00278
```

4.14.3.22 Constraint * ConstraintManager::getConstraint (int *id*) const

Returns the pointer to the constraint with id id'.

Definition at line 447 of file constraint_manager.C.

Referenced by AbsDistResiduals(), ConstraintManager(), InvalidateAtoms(), TurnAtomOn(), TurnAtomsOff(), TurnConstraintOn(), Constraint::assertValidity(), constraintON(), dStress(), setPosition(), and stress().

```
00448 {if(sG)
00449     return sG->GetData(id);
00450   else
00451
```

4.14.3.23 EnvelopeConstraint * ConstraintManager::getEnvelope () const [inline]

Definition at line 291 of file constraint_manager.h.

```
00292 {return EC;}
00293
```

4.14.3.24 const Molecule & ConstraintManager::getMolecule () const [inline]

Definition at line 274 of file constraint_manager.h.

Referenced by SatisfyMolecule::getMolecule(), and Constraint::getMolecule().

```
00274                                                   { return M; }
00275
```

4.14.3.25 DLList< Constraint *> ConstraintManager::getParents (int *Id*) const

Returns the parents of a constraint or atom. The parents are those constraints whose support contains a' or c'.

Definition at line 277 of file constraint_manager.C.

Referenced by getAssocAtoms().

```
00278 {
00279   //  makeStatic();
00280   DLList<Constraint*> ReturnList;
00281   if (sG){
00282
00283
00284     staticGraph::neighbor_iterator it;
00285     // cycle through our parents and collect pointers in list
00286     for(it = sG->begin_neighbor(Id,1);it; ++it)
00287       {
00288         ReturnList.AddToBack(sG->GetData(*it));
00289         // *it is an int
00290       }
00291     return ReturnList;
00292
00293   }
00294   else {
00295     DIE("getParents called on dynamic constraint graph");
00296     return ReturnList; // keep compiler happy
00297   }
00298
```

4.14.3.26 const jVector & ConstraintManager::getPosition (int *i*) const [inline]

Definition at line 278 of file constraint_manager.h.

Referenced by RandomPerturb(), TranslateModel(), molecule_center(), and operator()().

```
00278                                                          { return M(i); }
00279
```

4.14.3.27 double ConstraintManager::getRadius (int *i*) const [inline]

Definition at line 279 of file constraint_manager.h.

```
00279                                                  { return M.VDW(i); }
00280
```

4.14.3.28 jVector ConstraintManager::getStressList ()

Returns a **jVector** (p. 325) of the stress values; one for each atom.

Definition at line 152 of file constraint_manager.C.

```
00153 {
00154   int numCyc = M.Number();
00155   jVector StressList(numCyc);
00156   for (int i =0; i<numCyc; i++){
00157     StressList[i] = stress(i);
```

```
00158     //    cerr << "Label: " << getConstraint(i)->Label() << "STRESS: " << stress(i) << endl;
00159
00160   }
00161
```

4.14.3.29 double ConstraintManager::getWeight (int *id*) const [inline, virtual]

Definition at line 105 of file constraint_manager.h.

```
00105                                       {
00106     // not defn yet
00107     return 0;
00108
```

4.14.3.30 bool ConstraintManager::isDynamic () const [protected]

Returns true if the constraint graph is dynamic at this moment, and false if it is stored statically.

Definition at line 460 of file constraint_manager.C.

```
00461 { return (dG!=0); }
00462
```

4.14.3.31 bool ConstraintManager::makeDynamic () [protected]

Converts the graph from dynamic to static and vice verca. Returns true if successful and false otherwise.

Definition at line 479 of file constraint_manager.C.

Referenced by Constraint::Input(), add(), clearAllConstraints(), clearConstraints(), remove(), and setMolecule().

```
00480 {
00481   if(dG)
00482     return true;
00483   if(sG)
00484     {
00485       if(!(dG = new dynamicGraph(*sG)))
00486         return false;
00487       delete sG; sG = 0;
00488       return true;
00489     }
00490
```

4.14.3.32 bool ConstraintManager::makeStatic () [protected]

Converts the graph from dynamic to static and vice verca. Returns true if successful and false otherwise.

Definition at line 463 of file constraint_manager.C.

Referenced by PrintGraph(), Constraint::assertValidity(), Constraint::switchOff(), and ~ConstraintManager().

```
00464 {
00465   if(sG)
00466     return true;
00467
00468   if(!dG)
00469     return false;
00470
00471   if(!(sG = new staticGraph(*dG, true)))
00472     return false;
00473
00474   delete dG; dG = 0;
00475
00476   return true;
00477
```

4.14.3.33 jVector ConstraintManager::molecule_center (void) const

Definition at line 138 of file constraint_manager.C.

```
00139 {
00140   jVector rv(3,0);
00141   double m;
00142
00143   for(int i=numAtoms()-1; i>=0; i--){
00144     if (constraintON(i)){
00145       rv += getPosition(i);
00146       m++;
00147     }
00148   }
00149
```

4.14.3.34 DLList< int > ConstraintManager::neighbor_i (const jVector & *p*, double *distance*) const [inline]

Definition at line 284 of file constraint_manager.h.

```
00285 {return M.neighbor_i(p,distance); }
00286
```

4.14.3.35 int ConstraintManager::numAtoms () const [inline]

Definition at line 286 of file constraint_manager.h.

Referenced by AbsDistResiduals(), AnimFrame::AnimFrame(), InvalidateAtoms(), TranslateModel(), molecule_center(), Constraint::notifyOff(), numAtomsON(), numConstraints(), Constraint::stress(), and Constraint::switchOff().

```
00286                                          { return M.Number(); }
00287
```

4.14.3.36 int ConstraintManager::numAtomsON () const

Definition at line 545 of file constraint_manager.C.

```
00545                                       {
00546   int rv = 0;
00547   for (int i = numAtoms()-1;i>=0;i--){
00548     if (constraintON(i))
00549       rv++;
00550   }
00551
```

4.14.3.37 int ConstraintManager::numConstraints () const [inline]

Definition at line 61 of file constraint_manager.h.

```
00062     {
00063       return Number()-numAtoms();
00064
```

4.14.3.38 const jVector & ConstraintManager::operator() (int *i*) const [inline]

Definition at line 281 of file constraint_manager.h.

```
00281                                                          { return getPosition(i); }
00282
```

4.14.3.39 const Atom & ConstraintManager::operator[] (int *i*) const [inline]

Definition at line 280 of file constraint_manager.h.

```
00280                                                       { return getAtom(i); }
00281
```

4.14.3.40 void ConstraintManager::readNewMoleculeXmol (ifstream & *is*) [inline]

Definition at line 276 of file constraint_manager.h.

```
00276                                                        {M.readXmol(is);InvalidateAtoms();}
00277
```

4.14.3.41 void ConstraintManager::registerEnvelope (EnvelopeConstraint * *EC_in*) [inline]

Definition at line 289 of file constraint_manager.h.

```
00290 {if (EC) DIE("only 1 envelope allowed"); EC = EC_in;}
00291
```

4.14.3.42 void ConstraintManager::remove (Constraint & *c*) [protected]

Only friends (e.g. Constraints) and derived classes may add or remove themselves from the manager. The **add**() (p. 104) function returns the id assigned to the newly added constraint.

Definition at line 263 of file constraint_manager.C.

Referenced by Constraint::Input(), Constraint::InputBinary(), clearAllConstraints(), clearConstraints(), Constraint::notifyRemove(), and Constraint::setManager().

```
00264 {
00265   // Make sure this constraint has a valid ID:
00266   if(dG && !dG->HasVertex(&c))
00267     return;
00268   else if(sG && !sG->HasVertex(&c))
00269     return;
00270
00271   makeDynamic();               // Changing graph architecture, so need dynamic.
00272   c.notifyRemove();            // Notify the constraint it's being removed.
00273   // Recursively calls CM->remove() on all of c''s constraint support.
00274   dG->RemVertex(&c);    // Disconnect its node in the graph.
00275
```

4.14.3.43 void ConstraintManager::resetMoved (void) [inline]

set all atoms in molecule to 'not moved yet'.

Definition at line 287 of file constraint_manager.h.

```
00288 {for (int i = 0; i < M.Number(); i++) M[i].Moved()=false;}
00289
```

4.14.3.44 void ConstraintManager::setAtom (int *i*, const Atom & *a*) [inline]

Definition at line 117 of file constraint_manager.h.

```
00117                                     { M[i] = a; }
00118
```

4.14.3.45 void ConstraintManager::setMolecule (const Molecule & *mol*)

Definition at line 115 of file constraint_manager.C.

Referenced by SatisfyMolecule::setMolecule().

```
00116 {
00117   if(M.Number() != mol.Number())
00118     {
00119       DIE("Invalid new molecule!\n");
00120     }
00121
00122   M = mol;
00123   Constraint * constraint;
00124   if(makeDynamic())
00125     {
00126       int numCyc = dG->NumVertices();
00127       for(int i=M.Number(); i<=numCyc; i++)
```

```
00128         {
00129           constraint = dG->GetData(i);
00130           constraint->assertValidity(false);
00131         }
00132     }  else {
00133       DIE(" Error in setMolecule(...)");
00134     }
00135
```

4.14.3.46 void ConstraintManager::setPosition (int *i*, const jVector & *v*)

Definition at line 87 of file constraint_manager.C.

Referenced by RandomPerturb(), and TranslateModel().

```
00088 {
00089   if (i>M.Number()-1){
00090     DIE("tried to setPosition on non-Atom constraint");
00091   }
00092
00093   if (constraintON(i)){
00094
00095     // Set this constraint to be invalid when moved
00096     Constraint * C = getConstraint(i);
00097     C->assertValidity(false);
00098
00099     // Notify those constraints which depend on this atom
00100     // position that I have changed.
00101     Constraint * SuppC = 0;
00102     int numC = C->supportC.Length();
00103
00104     for (int j = 0;j<numC;j++){
00105       SuppC = getConstraint(C->supportC[j]);
00106       SuppC->notifyValid(C,false);
00107     }
00108     M.setPosition(i,v);
00109     M[i].Moved() = true; // set atom moved' to true
00110   } else {
00111     DIE ("setPosition called for atom that is off");
00112   }
00113
```

4.14.3.47 void ConstraintManager::setWeight (int *id*) [inline, virtual]

Definition at line 137 of file constraint_manager.h.

```
00137                                {
00138     // not defined yet
00139     return;
00140
```

4.14.3.48 double ConstraintManager::stress ()

Return sum of stress over all aloms in the molecule.

Definition at line 175 of file constraint_manager.C.

Referenced by gwNumerics::addLPERTSegments(), and getStressList().

```
00176 {
00177
00178   // calculates the stress of te whole molecule as the sum
00179
00180   double total = 0;
00181   for (int i =0; i<M.Number(); i++)
00182     total += stress(i);
00183   return total;
00184
```

4.14.3.49 double ConstraintManager::stress (int *atom*, const jVector & *position*) const

Return total stress for an atom as if it were in a new posistion.

Definition at line 170 of file constraint_manager.C.

```
00171 {
00172   return getConstraint(atom)->stress(atom,position) ;
00173
```

4.14.3.50 double ConstraintManager::stress (int *Id*)

Return stress total stress for an atom.

Definition at line 165 of file constraint_manager.C.

```
00166 {
00167   return getConstraint(Id)->stress();
00168
```

4.14.4 Friends And Related Function Documentation

4.14.4.1 class Constraint [friend]

Definition at line 48 of file constraint_manager.h.

4.14.4.2 class ConstraintRegistry [friend]

Definition at line 47 of file constraint_manager.h.

4.14.5 Member Data Documentation

4.14.5.1 AtomConstraint * ConstraintManager::AtomConstraints [protected]

Array of dummy constraints, each holding one atom in the graph.

Definition at line 224 of file constraint_manager.h.

4.14.5.2 EnvelopeConstraint * ConstraintManager::EC [protected]

//

Constraint (p. 80) Graphs:

Description:

There are 2 constraint graphs: one that is dynamic and one that is static. Only one of these graphs exists at any one point in time. When a constraint manager is constructed, it is given a new dynamic graph, dG, and the static graph, sG, is undefined. Friend classes will check whether the current graph of constraints is being stored statically or dynamically and act accordingly.

//

Definition at line 205 of file constraint_manager.h.

4.14.5.3 Molecule ConstraintManager::M [protected]

The single molecule this mgr works on.

Definition at line 260 of file constraint_manager.h.

Referenced by dStress(), and getPosition().

4.14.5.4 jVector ConstraintManager::TypeWeights [private]

Definition at line 269 of file constraint_manager.h.

4.14.5.5 dynamicGraph * ConstraintManager::dG [protected]

The dynamic constraint graph.

Definition at line 211 of file constraint_manager.h.

4.14.5.6 staticGraph * ConstraintManager::sG [protected]

The static constraint graph.

Definition at line 214 of file constraint_manager.h.

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

- **constraint_manager.h**
- **constraint_manager.C**

4.15 ConstraintRegistry Class Reference

`#include <constraint_registry.h>`

Static Public Methods

- **DynString getIdentifier** (int i=0)
- **Constraint*** **getConstraint** (const **DynString** &identifier, **ConstraintManager** &cm)
- int **readConstraints** (istream &is, **ConstraintManager** &cm, bool append=true)
- int **getIntType** (const **DynString** &identifier)

4.15.1 Detailed Description

//

File: contsraint_registry.h

The constraint registry generally interfaces the GRAPE system with real constraint types. It allows the reading of multiple different types of constraints and oganizes the correct creation of those constraints.

In the future I want the registry to facilitate run-time manipulation of constraint properties for a subset of constraints. This ability will require the calling function to have the ability to specify constraint types explicitly to the constraint manager. -jd

//

Definition at line 35 of file constraint_registry.h.

4.15.2 Member Function Documentation

4.15.2.1 Constraint * ConstraintRegistry::getConstraint (const DynString & *identifier*, ConstraintManager & *cm*) [static]

Definition at line 24 of file constraint_registry.C.

Referenced by readConstraints().

```
00026 {
00027
00028   // NB rigth now this is only called by the readConstraints command below.
00029   //
00030   Constraint * c = 0;
00031     // if none matches, then 0 & ignore.
00032     // if # matches, return -1, and skip rest of line.
00033
00034   if(identifier == DistanceConstraint::Identifier){
00035     c = new DistanceConstraint(cm);
00036     c->myType = DistanceConstraint::Identifier;
00037   }
00038   else if(identifier == AtomConstraint::Identifier){
00039     c = new AtomConstraint(cm);
00040     c->myType = AtomConstraint::Identifier;
00041     DIE("Atoms are being read during constraint addition.  Not Allowed");
00042   }
00043   else if(identifier == PositionConstraint::Identifier){
00044     c = new PositionConstraint(cm);
00045     c->myType = PositionConstraint::Identifier;
```

```
00046   }
00047   else if(identifier == EnvelopeConstraint::Identifier){
00048     c = new EnvelopeConstraint(cm);
00049     c->myType = EnvelopeConstraint::Identifier;
00050
00051   }
00052
```

4.15.2.2 DynString ConstraintRegistry::getIdentifier (int $i = 0$) [static]

4.15.2.3 int ConstraintRegistry::getIntType (const DynString & *identifier*) [static]

Definition at line 6 of file constraint_registry.C.

Referenced by ConstraintManager::AbsDistResiduals(), BinEnvelopeConstraint::PreSetup(), and getConstraint().

```
00006                                                                {
00007   // this is the ONLY place these numbers should be accessed
00008   if(identifier == AtomConstraint::Identifier)
00009     return 0;
00010   else if(identifier == DistanceConstraint::Identifier)
00011     return 1;
00012   else if(identifier == PositionConstraint::Identifier)
00013     return 2;
00014   else if(identifier == EnvelopeConstraint::Identifier)
00015     return 3;
00016   else if(identifier == BinEnvelopeConstraint::Identifier)
00017     return 4;
00018   else
00019     DIE("improper identifier");
00020
00021   return -1; // compiler warning
00022
```

4.15.2.4 int ConstraintRegistry::readConstraints (istream & *is*, ConstraintManager & *cm*, bool *append* = true) [static]

Definition at line 66 of file constraint_registry.C.

Referenced by main().

```
00068   {
00069     if(!append)
00070         cm.clearConstraints();
00071
00072     int start_num = cm.Number();
00073
00074     DynString constraintIdentifier;
00075     Constraint * c=0;
00076     //    while(!is.eof() && !is.bad())
00077     while(is.good())
00078       {
00079         // Read in the constraint identifier:
00080         //
00081         is >> constraintIdentifier;
00082
00083         // Given this identifer, retrieve the appropriate constraint
00084         // instance:
```

```
00085         //
00086         c = getConstraint(constraintIdentifier, cm);
00087
00088         // cout <<  " \"" << constraintIdentifier << "\"";
00089
00090         // Now that we have a particular type of constraint, we can
00091         // read it in:
00092         if(c > 0) {
00093           //cout << ",  read " <<  c->myType << endl;
00094           c->Input(is);  // The constraint manager "magically" logs this.
00095         }
00096       }
00097     if (cm.Number() - start_num > 0)
00098       return cm.Number();
00099     else {
00100       DIE ("Read 0 constaints from constraint file");
00101       return 0;
00102     }
00103
```

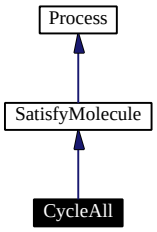
The documentation for this class was generated from the following files:

- **constraint_registry.h**
- **constraint_registry.C**

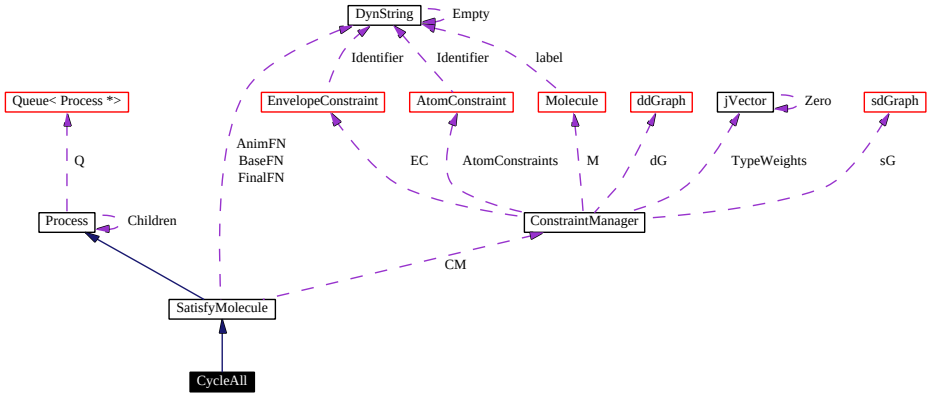
4.16 CycleAll Class Reference

`#include <cycle_all.h>`

Inheritance diagram for CycleAll



Collaboration diagram for CycleAll:



Public Methods

- **CycleAll** (const CycleAll &ca)
- **CycleAll** (**ConstraintManager** &cm)
- **~CycleAll** ()
- int **run** ()

4.16.1 Detailed Description

//

File: **cycle_all.h**

Created by: Jon Dugan

This is a 'job' that works in the GRAPE system. It touches each atom once and relaxes its position.

//

Definition at line 21 of file cycle_all.h.

4.16.2 Constructor & Destructor Documentation

4.16.2.1 CycleAll::CycleAll (const CycleAll & *ca*)

Definition at line 5 of file cycle_all.C.

```
00006   : SatisfyMolecule(*(ca.CM)){}
00007
```

4.16.2.2 CycleAll::CycleAll (ConstraintManager & *cm*)

Definition at line 9 of file cycle_all.C.

```
00010   : SatisfyMolecule(cm)
00011   {
00012     // Create one child process for each atom in the molecule:
00013     numC = CM->numAtoms();
00014     Children = new Process*[numC];
00015
00016     // Create a new relaxation process for every atom in the molecule:
00017     for(int i=0; i<numC; ++i)
00018       Children[i] = new gwRelaxAtom(*CM,i);
00019
00020
```

4.16.2.3 CycleAll::~CycleAll ()

Definition at line 22 of file cycle_all.C.

```
00023   {
00024     // Don't delete my children (since Process takes care of it).
00025     // i.e. dont do for(int i=0; i<numChildren; ++i) delete Children[i].
00026     // or delete[] Children.
00027
```

4.16.3 Member Function Documentation

4.16.3.1 int CycleAll::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 29 of file cycle_all.C.

```
00029                   {
00030 #ifdef JOB_VERBOSE
00031   cerr << "Start: \tCycleAll::run()" << endl;
00032 #endif
00033
00034     // save the largest current distance residual to statics
00035     CM->AbsDistResiduals(gwNumerics::MaxAbsDistRes,
00036                          gwNumerics::AveAbsDistRes);
00037
00038     // Force Envelope update next update
00039     CM->InvalidateEnvelope();
00040
```

```
00041
00042
00043 #ifdef JOB_VERBOSE2
00044         cerr << setprecision(50) << "ORDER:  \t" << "MaxAbsDistRes = "<< gwNumerics::MaxAbsDistRes << endl;
00045 #endif
00046
00047
00048   //      if(*maxcyclerr<0.0001) break;
00049   if (gwNumerics::MaxAbsDistRes < 0.0001)
00050     return 1; // stop now
00051
00052   // Compute the priorities of the children, based on their
00053   // residuals
00054   double tempPriority;
00055   for(int i=0; i<numC; ++i)
00056     {
00057       if (CM->constraintON(i)){
00058         tempPriority = CM->stress(i);
00059
00060 #ifdef JOB_VERBOSE2
00061         cerr << setprecision(50) << "Set ORDER:  \t" << "Priority for atom " <<i<<"="<< tempPriority << endl;
00062 #endif
00063
00064       Children[i]->setPriority(tempPriority);
00065
00066       }
00067     }
00068
00069   CM->resetMoved();
00070
00071     // Queue all the children to run queue
00072   //        for(int i=0; i<numC; ++i)    {
00073         for(int i=numC-1; i>=0; i--)    {
00074           if (CM->constraintON(i))
00075             addChild(Children[i]);
00076     }
00077
```

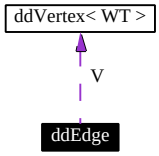
The documentation for this class was generated from the following files:

- **cycle_all.h**
- **cycle_all.C**

4.17 ddEdge Class Template Reference

`#include <ddvertex.h>`

Collaboration diagram for ddEdge:



Public Methods

- **ddEdge** ()
- **~ddEdge** ()
- int **Compare** (const ddEdge< WT > &e) const
- int **Input** (istream &is)
- int **Output** (ostream &os) const
- int **InputBinary** (istream &is)
- int **OutputBinary** (ostream &os) const
- **MAKECOMPARISONS_WITH_COMPARE** (ddEdge< WT >)
- **MAKE_IO_STREAM_OPERATORS** (ddEdge< WT >)
- **MAKE_BIN_STREAM_OPERATORS** (ddEdge< WT >)

Public Attributes

- **ddVertex**<WT>* **V**
- WT **W**

template<class WT = double> class ddEdge

4.17.1 Constructor & Destructor Documentation

4.17.1.1 template<class WT = double> ddEdge<WT>::ddEdge<WT> () [inline]

Definition at line 29 of file ddvertex.h.

```
00029                       { V=0; }
00030
```

4.17.1.2 template<class WT = double> ddEdge<WT>::~ddEdge<WT> () [inline]

Definition at line 30 of file ddvertex.h.

```
00030                        { /*empty*/ }
00031
```

4.17.2 Member Function Documentation

4.17.2.1 template<class WT> int ddEdge< WT >::Compare (const ddEdge< WT > & *e*) const [inline]

Definition at line 2 of file ddvertex.inc.

```
00003   { return (V->ID - e.V->ID); }
00004
```

4.17.2.2 template<class WT = double> int ddEdge<WT>::Input (istream & *is*) [inline]

Definition at line 39 of file ddvertex.h.

```
00040                 { is >> W; return 1; }
00041
```

4.17.2.3 template<class WT = double> int ddEdge<WT>::InputBinary (istream & *is*) [inline]

Definition at line 44 of file ddvertex.h.

```
00045                 { is >= W; return 1; }
00046
```

4.17.2.4 template<class WT = double> ddEdge<WT>::MAKECOMPARISONS_WITH_COMPARE (ddEdge< WT >)

4.17.2.5 template<class WT = double> ddEdge<WT>::MAKE_BIN_STREAM_OPERATORS (ddEdge< WT >)

4.17.2.6 template<class WT = double> ddEdge<WT>::MAKE_IO_STREAM_OPERATORS (ddEdge< WT >)

4.17.2.7 template<class WT = double> int ddEdge<WT>::Output (ostream & *os*) const [inline]

Definition at line 41 of file ddvertex.h.

```
00042                 { os << W; return 1; }
00043
```

4.17.2.8 template<class WT = double> int ddEdge<WT>::OutputBinary (ostream & *os*) const [inline]

Definition at line 46 of file ddvertex.h.

```
00047                 { os <= W; return 1; }
00048
```

4.17.3 Member Data Documentation

4.17.3.1 template<class WT = double> ddVertex< WT > * ddEdge<WT>::V

Definition at line 32 of file ddvertex.h.

4.17.3.2 template<class WT = double> WT ddEdge<WT>::W

Definition at line 34 of file ddvertex.h.

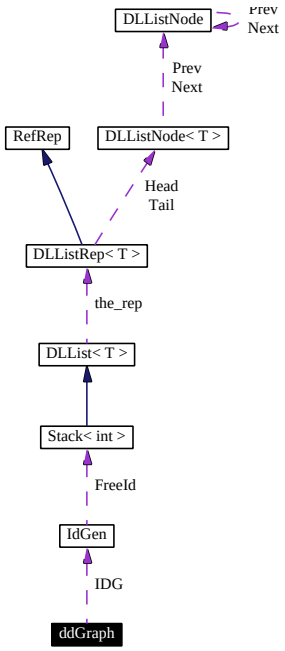
The documentation for this class was generated from the following files:

- **ddvertex.h**
- **ddvertex.inc**

4.18 ddGraph Class Template Reference

`#include <ddgraph.h>`

Collaboration diagram for ddGraph:



Public Types

- typedef ddGraph<DT,WT> **DDG**

Public Methods

- **ddGraph** ()
- **ddGraph** (const ddGraph< DT, WT > &ddg)
- **ddGraph** (const **sdGraph**< DT, WT > &sdg)
- **~ddGraph** ()
- void **Clear** ()
- void **Copy** (const ddGraph< DT, WT > &ddg)
- void **Copy** (const **sdGraph**< DT, WT > &ddg)
- int **NumVertices** () const
- int **NumEdges** () const
- int **HasVertex** (const DT &x) const
- int **HasVertex** (int n) const
- int **HasEdge** (const DT &from, const DT &to) const
- int **OutDegree** (const DT &x) const
- int **InDegree** (const DT &x) const
- int **Degree** (const DT &x) const
- int **NextFreeId** () const
- int **AddVertex** (const DT &x)

- **ddVertex**<WT>* **GetVertex** (const DT &x)
- int **GetVertexId** (const DT &x) const
- DT& **GetData** (int vid)
- const DT& **GetData** (int vid) const
- DT& **GetData** (const **ddVertex**< WT > *v)
- const DT& **GetData** (const **ddVertex**< WT > *v) const
- int **AddEdge** (const DT &from, const DT &to, const WT &fwd_weight, const WT &bwd_weight)
- int **AddEdge** (const DT &from, const DT &to, const WT &weight)
- int **AddEdge** (const DT &from, const DT &to)
- const WT& **GetWeight** (const DT &from, const DT &to, int fwd=1) const
- const WT& **operator()** (const DT &from, const DT &to) const
- void **SetWeight** (const DT &from, const DT &to, const WT &fwd_weight, const WT &bwd_weight)
- void **SetWeight** (const DT &from, const DT &to, const WT &weight)
- void **ChangeWeight** (const DT &from, const DT &to, const WT &fwd_del, const WT &bwd_del)
- void **ChangeWeight** (const DT &from, const DT &to, const WT &del)
- void **RemOutEdges** (const DT &x)
- void **RemInEdges** (const DT &x)
- void **RemEdges** (const DT &x)
- int **RemVertex** (const DT &x)
- int **RemEdge** (const DT &from, const DT &to)
- const ddGraph<DT,WT>& **operator=** (const ddGraph< DT, WT > &ddg)
- const ddGraph<DT,WT>& **operator=** (const **sdGraph**< DT, WT > &sdg)
- void **OutputSimple** (ostream &os)
- int **Output** (ostream &os) const
- int **Input** (istream &is)
- **MAKE_IO_STREAM_OPERATORS** (**DDG**)

Private Types

- typedef **ddEdge**<WT> **Edge**
- typedef **ddVertex**<WT> **Vertex**
- typedef map<DT, **Vertex***> **VertexArray**
- typedef map< int, DT > **DataArray**

Private Methods

- int **OutDegree** (const **ddVertex**< WT > *vert) const
- int **InDegree** (const **ddVertex**< WT > *vert) const
- int **Degree** (const **ddVertex**< WT > *vert) const
- int **AddEdge** (**ddVertex**< WT > *from, **ddVertex**< WT > *to, const WT &fwd_weight, const WT &bwd_weight)
- int **RemEdge** (**ddVertex**< WT > *from, **ddVertex**< WT > *to)
- void **SetWeight** (**ddVertex**< WT > *from, **ddVertex**< WT > *to, const WT &fwd_weight, const WT &bwd_weight)
- void **ChangeWeight** (**ddVertex**< WT > *from, **ddVertex**< WT > *to, const WT &fwd_del, const WT &bwd_del)

Private Attributes

- **IdGen IDG**
- int **NumV**
- int **NumE**
- **VertexArray VA**
- **DataArray Data**

Friends

- class **sdGraph< DT, WT >**
- istream& **InputBinary** (istream &is, ddGraph< DT, WT > &ddg)
- istream& **operator>=** (istream &is, ddGraph< DT, WT > &ddg)
- ostream& **OutputBinary** (ostream &os, const ddGraph< DT, WT > &ddg)
- ostream& **operator** (ostream &os, const ddGraph< DT, WT > &ddg)

template<class DT, class WT = double> class ddGraph

4.18.1 Member Typedef Documentation

4.18.1.1 template<class DT, class WT = double> typedef ddGraph< DT, WT > ddGraph<DT, WT>::DDG

Definition at line 167 of file ddgraph.h.

4.18.1.2 template<class DT, class WT = double> typedef map< int, DT > ddGraph<DT, WT>::DataArray [private]

Definition at line 191 of file ddgraph.h.

4.18.1.3 template<class DT, class WT = double> typedef ddEdge< WT > ddGraph<DT, WT>::Edge [private]

Definition at line 184 of file ddgraph.h.

4.18.1.4 template<class DT, class WT = double> typedef ddVertex< WT > ddGraph<DT, WT>::Vertex [private]

Definition at line 185 of file ddgraph.h.

4.18.1.5 template<class DT, class WT = double> typedef map< DT, Vertex *> ddGraph<DT, WT>::VertexArray [private]

Definition at line 186 of file ddgraph.h.

4.18.2 Constructor & Destructor Documentation

4.18.2.1 template<class DT, class WT> ddGraph< DT, WT >::ddGraph<DT, WT> () [inline]

Definition at line 132 of file ddgraph.inc.

```
00133   {
00134     NumV = 0;
00135     NumE = 0;
00136
```

4.18.2.2 template<class DT, class WT> ddGraph< DT, WT >::ddGraph<DT, WT> (const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 139 of file ddgraph.inc.

```
00140   {
00141     NumV = 0;
00142     NumE = 0;
00143     Copy(ddg);
00144
```

4.18.2.3 template<class DT, class WT> ddGraph< DT, WT >::ddGraph<DT, WT> (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 147 of file ddgraph.inc.

```
00148   {
00149     NumV = 0;
00150     NumE = 0;
00151     Copy(sdg);
00152
```

4.18.2.4 template<class DT, class WT> ddGraph< DT, WT >::~ddGraph<DT, WT> () [inline]

Definition at line 155 of file ddgraph.inc.

```
00156   { Clear(); }
00157
```

4.18.3 Member Function Documentation

4.18.3.1 template<class DT, class WT> int ddGraph< DT, WT >::AddEdge (ddVertex< WT > * *from*, ddVertex< WT > * *to*, const WT & *fwd_weight*, const WT & *bwd_weight*) [inline, private]

Definition at line 275 of file ddgraph.inc.

```
00277   {
00278     if(!from && !to)
00279         return VertexBad;
```

```
00280
00281     Edge eF, eT;
00282     eT.V = to;   eT.W = fwd_weight;
00283     eF.V = from; eF.W = bwd_weight;
00284
00285     // Check if the edge already exists
00286     if(from->OutE.count(eT)>0)
00287         return EdgeInG;
00288
00289     from->OutE.insert(eT);
00290     to->InE.insert(eF);
00291     return ++NumE;
00292
```

4.18.3.2 template<class DT, class WT> int ddGraph< DT, WT >::AddEdge (const DT & *from*, const DT & *to*) [inline]

Definition at line 317 of file ddgraph.inc.

```
00318   { WT weight; return AddEdge(from,to,weight,weight); }
00319
```

4.18.3.3 template<class DT, class WT> int ddGraph< DT, WT >::AddEdge (const DT & *from*, const DT & *to*, const WT & *weight*) [inline]

Definition at line 313 of file ddgraph.inc.

```
00314   { return AddEdge(from,to,weight,weight); }
00315
```

4.18.3.4 template<class DT, class WT> int ddGraph< DT, WT >::AddEdge (const DT & *from*, const DT & *to*, const WT & *fwd_weight*, const WT & *bwd_weight*) [inline]

Definition at line 295 of file ddgraph.inc.

Referenced by AddEdge(), Copy(), Input(), and ConstraintManager::add().

```
00297   {
00298     ddVertex<WT> * FromV, * ToV;
00299
00300     // If we dont have the vertices, just add them!
00301     //
00302     if((!HasVertex(from) && (AddVertex(from)<0)) ||
00303        (!HasVertex(to) && (AddVertex(to)<0)))
00304           return VertexAddFailure;
00305
00306     FromV = VA[from];
00307     ToV   = VA[to];
00308
00309     return AddEdge(FromV,ToV,fwd_weight,bwd_weight);
00310
```

4.18.3.5 template<class DT, class WT> int ddGraph< DT, WT >::AddVertex (const DT & *x*) [inline]

Definition at line 217 of file ddgraph.inc.

Referenced by AddEdge(), Copy(), and ConstraintManager::add().

```
00218   {
00219     if(HasVertex(x)) return VA[x]->ID;
00220     int id = IDG.Get();
00221     Data[id] = x;
00222     ddVertex<WT> * vert = new ddVertex<WT>(id);
00223     VA[x] = vert;
00224     ++NumV;
00225     return id;
00226
```

4.18.3.6 template<class DT, class WT> void ddGraph< DT, WT >::ChangeWeight (ddVertex< WT > * *from*, ddVertex< WT > * *to*, const WT & *fwd_del*, const WT & *bwd_del*) [inline, private]

Definition at line 367 of file ddgraph.inc.

```
00369   {
00370     if(!from || !to)
00371       { cerr << "ddGraph::ChangeWeight\n"; raise(SIGQUIT); }
00372
```

4.18.3.7 template<class DT, class WT> void ddGraph< DT, WT >::ChangeWeight (const DT & *from*, const DT & *to*, const WT & *delta*) [inline]

Definition at line 386 of file ddgraph.inc.

```
00388   { ChangeWeight(from,to,delta,delta); }
00389
```

4.18.3.8 template<class DT, class WT> void ddGraph< DT, WT >::ChangeWeight (const DT & *from*, const DT & *to*, const WT & *fwd_del*, const WT & *bwd_del*) [inline]

Definition at line 377 of file ddgraph.inc.

Referenced by ChangeWeight().

```
00379   {
00380     if(!HasEdge(from,to))
00381       { cerr << "ddGraph::ChangeWeight\n"; raise(SIGQUIT); }
00382
```

4.18.3.9 template<class DT, class WT> void ddGraph< DT, WT >::Clear () [inline]

Definition at line 55 of file ddgraph.inc.

Referenced by Copy(), Input(), and ~ddGraph().

```
00056   {
00057     for(VertexArray::iterator it = VA.begin(); it!=VA.end(); ++it)
00058         delete ($\ast$it).second;
00059     VA.clear();
00060     Data.clear();
00061     IDG.Reset();
00062     NumV = 0;
00063     NumE = 0;
00064
```

4.18.3.10 template<class DT, class WT> void ddGraph< DT, WT >::Copy (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 94 of file ddgraph.inc.

```
00095   {
00096     Clear();
00097
00098     // Add all the edges to the graph:
00099     //
00100     int k;
00101     sdEdge<WT> edge;
00102     edge.V = new sdVertex<WT>;
00103     for(int i=0; i<sdg.NumV; ++i)
00104       {
00105         if(AddVertex(sdg.Data[i])<0)
00106             { cerr << "ddGraph::Copy(sdGraph)#1\n"; raise(SIGQUIT); }
00107         edge.V->ID = i;
00108         for(int j=0; j<sdg.VA[i]->NumO; ++j)
00109           {
00110             k = BinSearch(sdg.VA[i]->OutE[j].V->InE,
00111                           sdg.VA[i]->OutE[j].V->NumI,
00112                           edge);
00113             if(k<0 || k>=sdg.VA[i]->OutE[j].V->NumI)
00114               { cerr << "ddGraph::Copy(sdGraph)#2\n"; raise(SIGQUIT); }
00115
00116             AddEdge(sdg.Data[i],
00117                     sdg.Data[(sdg.VA[i]->OutE[j].V->ID)],
00118                     sdg.VA[i]->OutE[j].W,
00119                     sdg.VA[i]->OutE[j].V->InE[k].W);
00120           }
00121       }
00122
```

4.18.3.11 template<class DT, class WT> void ddGraph< DT, WT >::Copy (const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 67 of file ddgraph.inc.

Referenced by ddGraph(), and operator=().

```
00068   {
00069     if(this == &ddg) return;
00070     Clear();
00071     if(ddg.NumV<=0) return;
00072
00073     // Add all the edges to the graph (this automatically takes care
00074     // of adding vertices btw.
00075     //
00076     set<Edge>::const_iterator eit;
```

```
00077     Edge edge;
00078     edge.V = new ddVertex<WT>;
00079     VertexArray::const_iterator vit;
00080     for(vit = ddg.VA.begin(); vit!=ddg.VA.end(); ++vit)
00081       {
00082         if((edge.V->ID = AddVertex((*vit).first))<0)
00083             { cerr << "ddGraph::Copy(ddGraph)\n"; raise(SIGQUIT); }
00084         for(eit = (($\ast$vit).second)->OutE.begin();
00085             eit!=(($\ast$vit).second)->OutE.end(); ++eit)
00086             AddEdge((*vit).first,
00087                     ddg.Data[($\ast$eit).V->ID],
00088                     eit->W, ((($\ast$eit).V)->InE).find(edge)->W);
00089       }
00090
```

4.18.3.12 template<class DT, class WT = double> int ddGraph<DT, WT>::Degree (const ddVertex< WT > * *vert*) const [private]

4.18.3.13 template<class DT, class WT> int ddGraph< DT, WT >::Degree (const DT & *x*) const [inline]

Definition at line 205 of file ddgraph.inc.

```
00206   {
00207     if(!HasVertex(x)) return VertexNotInG;
00208     ddVertex<WT>* vert = VA[x];
00209     return (vert->OutDegree() + vert->InDegree());
00210
```

4.18.3.14 template<class DT, class WT> const DT & ddGraph< DT, WT >::GetData (const ddVertex< WT > * *v*) const [inline]

Definition at line 267 of file ddgraph.inc.

```
00268   {
00269     if(!v)
00270       { cerr << "ddGraph::GetData(ddVertex<WT>*)\n"; raise(SIGQUIT); }
00271
```

4.18.3.15 template<class DT, class WT> DT & ddGraph< DT, WT >::GetData (const ddVertex< WT > * *v*) [inline]

Definition at line 259 of file ddgraph.inc.

```
00260   {
00261     if(!v)
00262       { cerr << "ddGraph::GetData(ddVertex<WT>*)\n"; raise(SIGQUIT); }
00263
```

4.18.3.16 template<class DT, class WT = double> const DT& ddGraph<DT, WT>::GetData (int *vid*) const

4.18.3.17 template<class DT, class WT> DT & ddGraph< DT, WT >::GetData (int *vid*) [inline]

Definition at line 243 of file ddgraph.inc.

Referenced by sdGraph::Copy(), GetData(), ConstraintManager::add(), ConstraintManager::clearAllConstraints(), ConstraintManager::clearConstraints(), ConstraintManager::getConstraint(), and ConstraintManager::setMolecule().

```
00244   {
00245     if(Data.count(vid)<1)
00246       { cerr << "ddGraph::GetData("<<vid<<")\n"; raise(SIGQUIT); }
00247
```

4.18.3.18 template<class DT, class WT> ddVertex< WT > * ddGraph< DT, WT >::GetVertex (const DT & *x*) [inline]

Definition at line 229 of file ddgraph.inc.

Referenced by sdGraph::Copy().

```
00230   {
00231     if(!HasVertex(x)) { cerr << "ddGraph::GetVertex\n"; raise(SIGQUIT); }
00232
```

4.18.3.19 template<class DT, class WT> int ddGraph< DT, WT >::GetVertexId (const DT & *x*) const [inline]

Definition at line 236 of file ddgraph.inc.

```
00237   {
00238     if(!HasVertex(x)) { cerr << "ddGraph::GetVertexId\n"; raise(SIGQUIT); }
00239
```

4.18.3.20 template<class DT, class WT> const WT & ddGraph< DT, WT >::GetWeight (const DT & *from*, const DT & *to*, int *fwd* = 1) const [inline]

Definition at line 321 of file ddgraph.inc.

Referenced by operator()().

```
00323   {
00324     Edge edge;
00325
00326     ddVertex<WT> * FromV, * ToV;
00327     if(!HasVertex(from) || !HasVertex(to))
00328       { cerr << "ddGraph::GetWeight\n"; raise(SIGQUIT); }
00329
00330     FromV = VA[from];
00331     ToV   = VA[to];
00332
00333     if(fwd)
00334       {
00335         edge.V = ToV;
00336         return FromV->OutE.Find(edge).W;
00337       }
00338
```

4.18.3.21 template<class DT, class WT> int ddGraph< DT, WT >::HasEdge (const DT & *from*, const DT & *to*) const [inline]

Definition at line 175 of file ddgraph.inc.

Referenced by ChangeWeight(), and SetWeight().

```
00176   {
00177     //
00178     // If either of the vertices is not in the graph then the edge must not be.
00179     //
00180     if(!HasVertex(from) || !HasVertex(to)) return 0;
00181
00182     ddVertex<WT> * FromV = VA[from], * ToV = VA[to];
00183     Edge eF; eF.V = FromV;
00184     Edge eT; eT.V = ToV;
00185     return (FromV->OutE.In(eT) && ToV->InE.In(eF));
00186
```

4.18.3.22 template<class DT, class WT> int ddGraph< DT, WT >::HasVertex (int *n*) const [inline]

Definition at line 171 of file ddgraph.inc.

```
00172   { return (Data.count(n)>0); }
00173
```

4.18.3.23 template<class DT, class WT> int ddGraph< DT, WT >::HasVertex (const DT & *x*) const [inline]

Definition at line 167 of file ddgraph.inc.

Referenced by AddEdge(), AddVertex(), sdGraph::Copy(), Degree(), GetVertex(), GetVertexId(), GetWeight(), HasEdge(), InDegree(), OutDegree(), RemEdge(), RemEdges(), RemInEdges(), RemOutEdges(), RemVertex(), and ConstraintManager::remove().

```
00168   { return (VA.count(x)>0); }
00169
```

4.18.3.24 template<class DT, class WT = double> int ddGraph<DT, WT>::InDegree (const ddVertex< WT > * *vert*) const [private]

4.18.3.25 template<class DT, class WT> int ddGraph< DT, WT >::InDegree (const DT & *x*) const [inline]

Definition at line 197 of file ddgraph.inc.

```
00198   {
00199     if(!HasVertex(x)) return VertexNotInG;
00200     ddVertex<WT>* vert = VA[x];
00201     return vert->InDegree();
00202
```

4.18.3.26 template<class DT, class WT> int ddGraph< DT, WT >::Input (istream & *is*) [inline]

Definition at line 512 of file ddgraph.inc.

```
00513   {
00514     //
00515     int id, from_id, to_id;
00516     DT data;
00517     WT weight;
00518     ddVertex<WT> * new_v, * from, * to;
00519     VertexArray::iterator it;
00520     DynString str;
00521
00522     map<int,int> IdToOcc;
00523         // Keeps track of vertex Id occurrences.
00524
00525     map<int, ddVertex<WT>* > IdToPtr;
00526         // Keeps a mapping of the id's to their vertices.
00527
00528     // First we need to delete everything in the graph.
00529     Clear();
00530
00531     int error = 0;
00532     int re_id = 0;
00533
00534     // Skip any space before the beginning vertex delimiter:
00535     for(is >> str; str != V_BEGIN_DELIM; is >> str);
00536
00537     // Read in the vertices until another delimiter is encountered:
00538     for(is >> str; str != V_END_DELIM; is >> str)
00539       {
00540         // The string should now contain an id and the next information
00541         // in the stream should contain the data item:
00542         //
00543         id = str.ToInt();
00544         if(is.eof() || is.bad()) { error = 1; break; }
00545         is >> data;
00546
00547         if(id < 0) re_id = 1;
00548
00549         // See if this id has already been used.  If it hasnt, record that
00550         // we've seen it, then create it and place it in the array.
00551         //
00552         if(IdToOcc.count(id)>0) { error = 1; break; }
00553         IdToOcc[id] = NumV;
00554         new_v = new ddVertex<WT>(id);
00555
00556         Data[id] = data;
00557         VA[data] = new_v;
00558         IdToPtr[id] = new_v;
00559
00560         ++NumV;
00561       }
00562
00563     // Read-in the edges.
00564     //
00565     int done = 0;
00566     if(!error)
00567       {
00568         // Again, skip any space before the edge delimiter is encountered:
00569         for(is >> str; str != E_BEGIN_DELIM; is >> str);
00570
00571         while(1)
```

```
00572           {
00573             for(int i=0; i<3; ++i)
00574               {
00575                 if(is.eof() || is.bad()) done = 1;
00576                 is >> str;
00577                 if((str == E_END_DELIM) || done)
00578                   {
00579                     done = 1;
00580                     if(i != 0) error = 1;
00581                     break;
00582                   }
00583                 if(i == 0)      from_id = str.ToInt();  // get first v's id.
00584                 else if(i == 1) to_id   = str.ToInt();  // get secnd v's id.
00585                 else            weight  = str.ToDbl();  // get the weight.
00586               }
00587              if(error || done) break;
00588
00589             // Check if the id's are valid.
00590             if(IdToPtr.count(from_id)<1 || IdToPtr.count(to_id)<1)
00591               { error = 1; break; }
00592
00593             from = IdToPtr[from_id];
00594             to   = IdToPtr[to_id];
00595
00596             if(AddEdge(from,to,weight,weight) < 0)
00597                 { error = 1; break; }
00598           }
00599
00600         // Re-assign the ID's if bad ID's were encountered.
00601         //
00602         if(!error)
00603           {
00604             if(re_id)
00605               {
00606                 for(it = VA.begin(); it!=VA.end(); ++it)
00607                     (($\ast$it).second)->ID = IdToOcc[(($\ast$it).second)->ID];
00608               }
00609             return 1;
00610           }
00611         //
00612       }
00613
```

4.18.3.27 template<class DT, class WT = double> ddGraph<DT, WT>::MAKE_IO_STREAM_OPERATORS (DDG)

4.18.3.28 template<class DT, class WT> int ddGraph< DT, WT >::NextFreeId () const [inline]

Definition at line 213 of file ddgraph.inc.

```
00214   { return IDG.Peek(); }
00215
```

4.18.3.29 template<class DT, class WT> int ddGraph< DT, WT >::NumEdges () const [inline]

Definition at line 163 of file ddgraph.inc.

```
00164   { return NumE; }
00165
```

4.18.3.30 template<class DT, class WT> int ddGraph< DT, WT >::NumVertices () const [inline]

Definition at line 159 of file ddgraph.inc.

Referenced by ConstraintManager::Number(), ConstraintManager::clearAllConstraints(), ConstraintManager::clearConstraints(), and ConstraintManager::setMolecule().

```
00160   { return NumV; }
00161
```

4.18.3.31 template<class DT, class WT = double> int ddGraph<DT, WT>::OutDegree (const ddVertex< WT > * *vert*) const [private]

4.18.3.32 template<class DT, class WT> int ddGraph< DT, WT >::OutDegree (const DT & *x*) const [inline]

Definition at line 189 of file ddgraph.inc.

```
00190   {
00191     if(!HasVertex(x)) return VertexNotInG;
00192     ddVertex<WT>* vert = VA[x];
00193     return vert->OutDegree();
00194
```

4.18.3.33 template<class DT, class WT> int ddGraph< DT, WT >::Output (ostream & *os*) const [inline]

Definition at line 489 of file ddgraph.inc.

```
00490   {
00491     VertexArray::const_iterator it;
00492     // Print the ID and data of all the vertices:
00493     //
00494     os << V_BEGIN_DELIM << '\n';
00495     for(it = VA.begin(); it!=VA.end(); ++it)
00496         os << '\t' << ((*it).second)->ID << '\t' << ($\ast$it).first << '\n';
00497     os << V_END_DELIM << '\n';
00498
00499     // Print the edges of the vertices
00500     //
00501     os << E_BEGIN_DELIM << '\n';
00502     for(it = VA.begin(); it!=VA.end(); ++it)
00503       {
00504         if(($\ast$it).second) (($\ast$it).second)->PrintOutEdges(os);
00505       }
00506
```

4.18.3.34 template<class DT, class WT = double> void ddGraph<DT, WT>::OutputSimple (ostream & *os*)

4.18.3.35 template<class DT, class WT> int ddGraph< DT, WT >::RemEdge (ddVertex< WT > * *from*, ddVertex< WT > * *to*) [inline, private]

Definition at line 458 of file ddgraph.inc.

```
00459   {
00460     if(NumE<=0) return NoEdges;
00461     if(!from || !to)
00462         return VertexBad;
00463
00464     Edge eF, eT;
00465     eF.V = from; eT.V = to;
00466
00467     from->OutE.Remove(eT);
00468     to->InE.Remove(eF);
00469     return --NumE;
00470
```

4.18.3.36 template<class DT, class WT> int ddGraph< DT, WT >::RemEdge (const DT & *from*, const DT & *to*) [inline]

Definition at line 473 of file ddgraph.inc.

```
00474   {
00475     ddVertex<WT> * FromV, * ToV;
00476
00477     if(NumV<=0) return NoVertices;
00478     if(NumE<=0) return NoEdges;
00479
00480     if(!HasVertex(from) || !HasVertex(to))
00481         return VertexNotInG;
00482
00483     FromV = VA[from];
00484     ToV = VA[to];
00485     return RemEdge(FromV,ToV);
00486
```

4.18.3.37 template<class DT, class WT> void ddGraph< DT, WT >::RemEdges (const DT & *x*) [inline]

Definition at line 413 of file ddgraph.inc.

```
00414   {
00415     if(!HasVertex(x))
00416       { cerr << "ddGraph::Remdges\n"; raise(SIGQUIT); }
00417
```

4.18.3.38 template<class DT, class WT> void ddGraph< DT, WT >::RemInEdges (const DT & *x*) [inline]

Definition at line 404 of file ddgraph.inc.

```
00405   {
00406     if(!HasVertex(x))
00407       { cerr << "ddGraph::RemInEdges\n"; raise(SIGQUIT); }
00408
```

4.18.3.39 template<class DT, class WT> void ddGraph< DT, WT >::RemOutEdges (const DT & x) [inline]

Definition at line 395 of file ddgraph.inc.

```
00396   {
00397     if(!HasVertex(x))
00398       { cerr << "ddGraph::RemOutEdges\n"; raise(SIGQUIT); }
00399
```

4.18.3.40 template<class DT, class WT> int ddGraph< DT, WT >::RemVertex (const DT & x) [inline]

Definition at line 422 of file ddgraph.inc.

Referenced by ConstraintManager::remove().

```
00423   {
00424     if(NumV <= 0) return NoVertices;
00425
00426     // ddVertex<WT> doesn't exist in the graph.
00427     if(!HasVertex(x))
00428         return VertexNotInG;
00429
00430     ddVertex<WT>* vert = VA[x];
00431     if(!vert) return VertexBad;
00432
00433     //
00434     // Clear all its edges and the edges it shares with other vertices.
00435     //
00436     NumE -= vert->Degree();
00437
00438     //
00439     // Remove the vertex from the array.
00440     //
00441     VA.erase(x);
00442     Data.erase(vert->ID);
00443
00444     //
00445     // Recycle this vertex's ID
00446     //
00447     IDG.Free(vert->ID);
00448
00449     //
00450     // And finally, delete the vertex from existence.
00451     //
00452     delete vert;
00453
00454     return --NumV;
00455
```

4.18.3.41 template<class DT, class WT> void ddGraph< DT, WT >::SetWeight (ddVertex< WT > * *from*, ddVertex< WT > * *to*, const WT & *fwd_weight*, const WT & *bwd_weight*) [inline, private]

Definition at line 343 of file ddgraph.inc.

```
00345   {
00346     if(!from || !to)
00347       { cerr << "ddGraph::SetWeight\n"; raise(SIGQUIT); }
00348
```

4.18.3.42 template<class DT, class WT> void ddGraph< DT, WT >::SetWeight (const DT & *from*, const DT & *to*, const WT & *weight*) [inline]

Definition at line 362 of file ddgraph.inc.

```
00364   { SetWeight(from,to,weight,weight); }
00365
```

4.18.3.43 template<class DT, class WT> void ddGraph< DT, WT >::SetWeight (const DT & *from*, const DT & *to*, const WT & *fwd_weight*, const WT & *bwd_weight*) [inline]

Definition at line 353 of file ddgraph.inc.

Referenced by SetWeight().

```
00355   {
00356     if(!HasEdge(from,to))
00357       { cerr << "ddGraph::SetWeight\n"; raise(SIGQUIT); }
00358
```

4.18.3.44 template<class DT, class WT> const WT & ddGraph< DT, WT >::operator() (const DT & *from*, const DT & *to*) const [inline]

Definition at line 391 of file ddgraph.inc.

```
00392   { return GetWeight(from,to); }
00393
```

4.18.3.45 template<class DT, class WT> const ddGraph< DT, WT > & ddGraph< DT, WT >::operator= (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 627 of file ddgraph.inc.

```
00628   { Copy(sdg); return *this; }
00629
```

4.18.3.46 template<class DT, class WT> const ddGraph< DT, WT > & ddGraph< DT, WT >::operator= (const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 623 of file ddgraph.inc.

```
00624   { Copy(ddg); return *this; }
00625
```

4.18.4 Friends And Related Function Documentation

4.18.4.1 template<class DT, class WT = double> istream& InputBinary (istream & *is*, ddGraph< DT, WT > & *ddg*) [friend]

Definition at line 632 of file ddgraph.inc.

```
00633   {
00634     sdGraph<DT,WT> G;
00635     if(InputBinary(is,G).bad())
00636         return is;
00637     ddg = G;
00638     return is;
00639
```

4.18.4.2 template<class DT, class WT = double> ostream& OutputBinary (ostream & *os*, const ddGraph< DT, WT > & *ddg*) [friend]

Definition at line 642 of file ddgraph.inc.

```
00643   {
00644     sdGraph<DT,WT> G(ddg);
00645     return OutputBinary(os,G);
00646
```

4.18.4.3 template<class DT, class WT = double> ostream& operator (ostream & *os*, const ddGraph< DT, WT > & *ddg*) [friend]

4.18.4.4 template<class DT, class WT = double> istream& operator>= (istream & *is*, ddGraph< DT, WT > & *ddg*) [friend]

Definition at line 649 of file ddgraph.inc.

```
00650   { return IntputBinary(is,ddg); }
00651
```

4.18.4.5 template<class DT, class WT = double> class sdGraph [friend]

Definition at line 73 of file ddgraph.h.

4.18.5 Member Data Documentation

4.18.5.1 template<class DT, class WT = double> DataArray ddGraph<DT, WT>::Data [private]

Definition at line 192 of file ddgraph.h.

4.18.5.2 template<class DT, class WT = double> IdGen ddGraph<DT, WT>::IDG [private]

Definition at line 177 of file ddgraph.h.

4.18.5.3 template<class DT, class WT = double> int ddGraph<DT, WT>::NumE [private]

Definition at line 181 of file ddgraph.h.

4.18.5.4 template<class DT, class WT = double> int ddGraph<DT, WT>::NumV [private]

Definition at line 179 of file ddgraph.h.

4.18.5.5 template<class DT, class WT = double> VertexArray ddGraph<DT, WT>::VA [private]

Definition at line 187 of file ddgraph.h.

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

- **ddgraph.h**
- **ddgraph.inc**

4.19 ddVertex Class Template Reference

`#include <ddvertex.h>`

Public Types

- typedef set<**ddEdge**<WT> > **EdgeSet**

Public Methods

- **ddVertex** (int id)
- **ddVertex** ()
- **~ddVertex** ()
- int **HaveSelfLoop** () const
- int **OutDegree** () const
- int **InDegree** () const
- int **Degree** () const
- const double& **GetWeightTo** (ddVertex< WT > *v) const
- const double& **GetWeightFrom** (ddVertex< WT > *v) const
- void **SetWeightTo** (ddVertex< WT > *v, const WT &weight)
- void **SetWeightFrom** (ddVertex< WT > *v, const WT &weight)
- void **ChangeWeightTo** (ddVertex< WT > *v, const WT &delta)
- void **ChangeWeightFrom** (ddVertex< WT > *v, const WT &delta)
- int **RemOutEdges** ()
- int **RemInEdges** ()
- int **RemEdges** ()
- void **PrintOutEdges** (ostream &os) const
- void **PrintInEdges** (ostream &os) const
- void **PrintEdges** (ostream &os) const
- int **Compare** (const ddVertex< WT > &v) const
- **MAKECOMPARISONS_WITH_COMPARE** (ddVertex< WT >)

Public Attributes

- int **ID**
- **EdgeSet OutE**
- **EdgeSet InE**

Friends

- ostream& **operator** (ostream &os, const ddVertex< WT > &vert)

template<class WT = double> class ddVertex

4.19.1 Member Typedef Documentation

4.19.1.1 template<class WT = double> typedef set< ddEdge< WT > > ddVertex<WT>::EdgeSet

Definition at line 68 of file ddvertex.h.

4.19.2 Constructor & Destructor Documentation

4.19.2.1 template<class WT> ddVertex< WT >::ddVertex<WT> (int *id*) [inline]

Definition at line 12 of file ddvertex.inc.

```
00013   {
00014     ID = id>=0 ? id : -1;
00015
```

4.19.2.2 template<class WT> ddVertex< WT >::ddVertex<WT> () [inline]

Definition at line 18 of file ddvertex.inc.

```
00019   {
00020     ID = -1;
00021
```

4.19.2.3 template<class WT> ddVertex< WT >::~ddVertex<WT> () [inline]

Definition at line 177 of file ddvertex.inc.

```
00178   {
00179     RemEdges();
00180
```

4.19.3 Member Function Documentation

4.19.3.1 template<class WT> void ddVertex< WT >::ChangeWeightFrom (ddVertex< WT > * *v*, const WT & *delta*) [inline]

Definition at line 85 of file ddvertex.inc.

Referenced by ddGraph::ChangeWeight().

```
00086   {
00087     ddEdge<WT> e; e.V = v;
00088     if(!v || InE.count(e)<1) raise(SIGQUIT);
00089     ((ddEdge<WT>&)(*(InE.find(e)))).W += delta;
00090
```

4.19.3.2 template<class WT> void ddVertex< WT >::ChangeWeightTo (ddVertex< WT > * *v*, const WT & *delta*) [inline]

Definition at line 77 of file ddvertex.inc.

Referenced by ddGraph::ChangeWeight().

```
00078   {
00079     ddEdge<WT> e; e.V = v;
00080     if(!v || OutE.count(e)<1) raise(SIGQUIT);
00081     ((ddEdge<WT>&)(*(OutE.find(e)))).W += delta;
00082
```

4.19.3.3 template<class WT = double> int ddVertex<WT>::Compare (const ddVertex< WT > & v) const [inline]

Definition at line 100 of file ddvertex.h.

```
00101                   { return (ID - v.ID); }
00102
```

4.19.3.4 template<class WT> int ddVertex< WT >::Degree () const [inline]

Definition at line 41 of file ddvertex.inc.

Referenced by ddGraph::RemVertex().

```
00042   { return OutDegree() + InDegree() - HaveSelfLoop(); }
00043
```

4.19.3.5 template<class WT> const double & ddVertex< WT >::GetWeightFrom (ddVertex< WT > * v) const [inline]

Definition at line 53 of file ddvertex.inc.

```
00054   {
00055     ddEdge<WT> e; e.V = v;
00056     if(!v || InE.count(e)<1) raise(SIGQUIT);
00057     return (*(InE.find(e))).W;
00058
```

4.19.3.6 template<class WT> const double & ddVertex< WT >::GetWeightTo (ddVertex< WT > * v) const [inline]

Definition at line 45 of file ddvertex.inc.

```
00046   {
00047     ddEdge<WT> e; e.V = v;
00048     if(!v || OutE.count(e)<1) raise(SIGQUIT);
00049     return (*(OutE.find(e))).W;
00050
```

4.19.3.7 template<class WT> int ddVertex< WT >::HaveSelfLoop () const [inline]

Definition at line 24 of file ddvertex.inc.

Referenced by Degree().

```
00025   {
00026     if(OutE.size()==0) return 0;
00027     for(EdgeSet::const_iterator it = OutE.begin(); it!=OutE.end(); ++it)
00028         if(($\ast$it).V == this) return 1;
00029     return 0;
00030
```

4.19.3.8 template<class WT> int ddVertex< WT >::InDegree () const [inline]

Definition at line 37 of file ddvertex.inc.

Referenced by sdGraph::Copy(), Degree(), ddGraph::Degree(), and ddGraph::InDegree().

```
00038   { return InE.size(); }
00039
```

4.19.3.9 template<class WT = double> ddVertex<WT>::MAKECOMPARISONS_WITH_COMPARE (ddVertex< WT >)

4.19.3.10 template<class WT> int ddVertex< WT >::OutDegree () const [inline]

Definition at line 33 of file ddvertex.inc.

Referenced by sdGraph::Copy(), Degree(), ddGraph::Degree(), and ddGraph::OutDegree().

```
00034   { return OutE.size(); }
00035
```

4.19.3.11 template<class WT> void ddVertex< WT >::PrintEdges (ostream & *os*) const [inline]

Definition at line 117 of file ddvertex.inc.

```
00118   { PrintOutEdges(os); PrintInEdges(os); }
00119
```

4.19.3.12 template<class WT> void ddVertex< WT >::PrintInEdges (ostream & *os*) const [inline]

Definition at line 105 of file ddvertex.inc.

Referenced by PrintEdges().

```
00106   {
00107     EdgeSet::const_iterator it;
00108     ddEdge<WT> edge;
00109     for(it = InE.begin(); it!=InE.end(); ++it)
00110       {
00111         edge = *it;
00112         os << '\t' << edge.V->ID << '\t' << ID << '\t' << edge.W << '\n';
00113       }
00114
```

4.19.3.13 template<class WT> void ddVertex< WT >::PrintOutEdges (ostream & *os*) const [inline]

Definition at line 93 of file ddvertex.inc.

Referenced by PrintEdges().

```
00094   {
00095     EdgeSet::const_iterator it;
00096     ddEdge<WT> edge;
00097     for(it = OutE.begin(); it!=OutE.end(); ++it)
00098       {
00099         edge = *it;
00100         os << '\t' << ID << '\t' << edge.V->ID << '\t' << edge.W << '\n';
00101       }
00102
```

4.19.3.14 template<class WT> int ddVertex< WT >::RemEdges () [inline]

Definition at line 173 of file ddvertex.inc.
Referenced by ddGraph::RemEdges(), and ~ddVertex().

```
00174   { return RemOutEdges() + RemInEdges(); }
00175
```

4.19.3.15 template<class WT> int ddVertex< WT >::RemInEdges () [inline]

Definition at line 155 of file ddvertex.inc.
Referenced by RemEdges(), and ddGraph::RemInEdges().

```
00156   {
00157     ddEdge<WT> t;
00158     int n = InE.size();
00159     t.V = this;
00160     // Remove all our neighbors' outgoing edges to us.
00161     //
00162     for(EdgeSet::iterator it = InE.begin(); it!=InE.end(); ++it)
00163         if(($\ast$it).V)
00164             ($\ast$it).V->OutE.erase(t);
00165
00166     // Remove all our incoming edges.
00167     //
00168     InE.clear();
00169     return n;
00170
```

4.19.3.16 template<class WT> int ddVertex< WT >::RemOutEdges () [inline]

Definition at line 132 of file ddvertex.inc.
Referenced by RemEdges(), and ddGraph::RemOutEdges().

```
00133   {
00134     //
00135     // This is a one sided removal!  Otherwise it may cause the IterateOver()
00136     // function to break: we can't remove the edges from the array we've
00137     // been called from, just those from the vertices we point to.
00138     //
00139     ddEdge<WT> f;
00140     int n = OutE.size();
00141     f.V = this;
00142     // Remove all our neighbors' incoming edges from us.
00143     //
```

```
00144     for(EdgeSet::iterator it = OutE.begin(); it!=OutE.end(); ++it)
00145         if(($\ast$it).V)
00146             ($\ast$it).V->InE.erase(f);
00147
00148     // Remove all our edges.
00149     //
00150     OutE.clear();
00151     return n;
00152
```

4.19.3.17 template<class WT> void ddVertex< WT >::SetWeightFrom (ddVertex< WT > * *v*, const WT & *weight*) [inline]

Definition at line 69 of file ddvertex.inc.

Referenced by ddGraph::SetWeight().

```
00070   {
00071     ddEdge<WT> e; e.V = v;
00072     if(!v || InE.count(e)<1) raise(SIGQUIT);
00073     ((ddEdge<WT>&)(*(InE.find(e)))).W = weight;
00074
```

4.19.3.18 template<class WT> void ddVertex< WT >::SetWeightTo (ddVertex< WT > * *v*, const WT & *weight*) [inline]

Definition at line 61 of file ddvertex.inc.

Referenced by ddGraph::SetWeight().

```
00062   {
00063     ddEdge<WT> e; e.V = v;
00064     if(!v || OutE.count(e)<1) raise(SIGQUIT);
00065     ((ddEdge<WT>&)(*(OutE.find(e)))).W = weight;
00066
```

4.19.4 Friends And Related Function Documentation

4.19.4.1 template<class WT = double> ostream& operator (ostream & *os*, const ddVertex< WT > & *vert*) [friend]

4.19.5 Member Data Documentation

4.19.5.1 template<class WT = double> int ddVertex<WT>::ID

Definition at line 65 of file ddvertex.h.

4.19.5.2 template<class WT = double> EdgeSet ddVertex<WT>::InE

Definition at line 70 of file ddvertex.h.

4.19.5.3 template<class WT = double> EdgeSet ddVertex<WT>::OutE

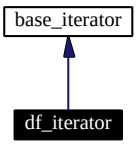
Definition at line 69 of file ddvertex.h.

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

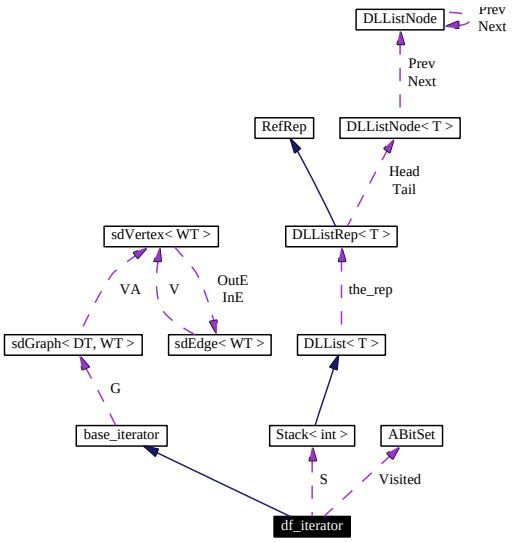
- **ddvertex.h**
- **ddvertex.inc**

4.20 df_iterator Class Reference

Inheritance diagram for df_iterator



Collaboration diagram for df_iterator:



Public Methods

- **df_iterator** (int reverse=0)
- **df_iterator** (const df_iterator &dfi)
- **~df_iterator** ()
- const df_iterator& **operator=** (const df_iterator &dfi)
- void **reset** ()
- **operator int** () const
- const **base_iterator**& **operator++** ()
- const **base_iterator**& **operator–** ()
- int **operator** * () const

Protected Methods

- **df_iterator** (**sdGraph**< DT, WT > *g, int vndx, int reverse=0)

Private Attributes

- **ABS* Visited**
- **Stack**<int> **S**

Friends

- class **sdGraph< DT, WT >**

4.20.1 Constructor & Destructor Documentation

4.20.1.1 df_iterator::df_iterator (sdGraph< DT, WT > * *g*, int *vndx*, int *reverse* = 0) [inline, protected]

Definition at line 114 of file sdgiter.inc.

```
00115                           : base_iterator(g,vndx,reverse)
00116
```

4.20.1.2 df_iterator::df_iterator (int *reverse* = 0) [inline]

Definition at line 119 of file sdgiter.inc.

```
00119                                       : base_iterator(reverse)
00120
```

4.20.1.3 df_iterator::df_iterator (const df_iterator & *dfi*) [inline]

Definition at line 123 of file sdgiter.inc.

```
00123                                                 : base_iterator(dfi)
00124                   {
00125                     Visited = new ABS(*(dfi.Visited));
00126                     S = dfi.S;
00127
```

4.20.1.4 df_iterator::~df_iterator () [inline]

Definition at line 129 of file sdgiter.inc.

```
00130                   { delete Visited; }
00131
```

4.20.2 Member Function Documentation

4.20.2.1 int df_iterator::operator * () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 204 of file sdgiter.inc.

```
00204                                     { return vid; }
00205
```

4.20.2.2 df_iterator::operator int () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 156 of file sdgiter.inc.

```
00157                 { return (vid!=-1); }
00158
```

4.20.2.3 const base_iterator & df_iterator::operator++ () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 159 of file sdgiter.inc.

```
00160                 {
00161                   int visited, nbrid;
00162
00163                   // We can stop if we've visited all the vertices:
00164                   if(Visited->IsAllSet()) vid = -1;
00165                   if(vid==-1) return *this;
00166
00167                   const sdEdge<WT>* edges = rev ? G->VA[vid]->InE :
00168                                                   G->VA[vid]->OutE;
00169                   int n = rev ? G->VA[vid]->NumI : G->VA[vid]->NumO;
00170
00171                   // Push all the unvisited neighbors onto the stack:
00172                   for(int i=0; i<n; ++i)
00173                     {
00174                       nbrid = edges[i].V->ID;
00175                       if(!Visited->IsSet(nbrid))
00176                           S.Push(nbrid);
00177                     }
00178
00179                   // Skip all the visited vertices in the stack and
00180                   // assign 'vid' to the first unvisited vertex:
00181                   vid = -1;
00182                   visited = 1;
00183                   while(!S.IsEmpty() && visited)
00184                     {
00185                       vid = S.Pop();
00186                       if((visited = Visited->IsSet((uint) vid)))
00187                           vid = -1;
00188                     }
00189
```

4.20.2.4 const base_iterator & df_iterator::operator– () [inline]

Definition at line 196 of file sdgiter.inc.

```
00197                 {
00198                   rev = rev ? 0 : 1;
00199                   ++($\ast$this);
00200                   rev = rev ? 0 : 1;
00201                   return *this;
00202
```

4.20.2.5 const df_iterator & df_iterator::operator= (const df_iterator & *dfi*) [inline]

Definition at line 132 of file sdgiter.inc.

```
00133                 {
00134                   if(this!=&dfi)
00135                     {
00136                       G=dfi.G; vstart=dfi.vstart; vid=dfi.vid; rev=dfi.rev;
00137                       S = dfi.S;
00138                       delete Visited;
00139                       Visited = new ABS(*(dfi.Visited));
00140                     }
00141
```

4.20.2.6 void df_iterator::reset () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).

Definition at line 144 of file sdgiter.inc.

```
00145                 {
00146                   if(bad()) vid=-1;
00147                   else
00148                     {
00149                       S.Clear();
00150                       vid = vstart;
00151                       Visited->ClearAll();
00152                       Visited->Set((uint) vid);
00153                     }
00154
```

4.20.3 Friends And Related Function Documentation

4.20.3.1 class sdGraph [friend]

Reimplemented from **base_iterator** (p. 66).

Definition at line 111 of file sdgiter.inc.

4.20.4 Member Data Documentation

4.20.4.1 Stack< int > df_iterator::S [private]

Definition at line 211 of file sdgiter.inc.

4.20.4.2 ABS * df_iterator::Visited [private]

Definition at line 208 of file sdgiter.inc.

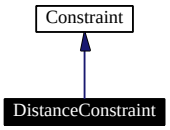
The documentation for this class was generated from the following file:

- **sdgiter.inc**

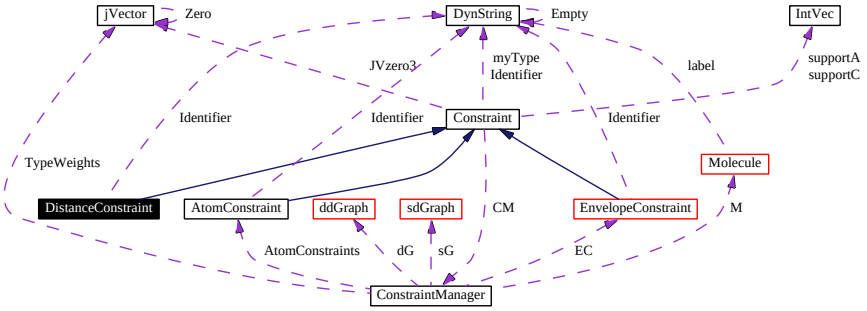
4.21 DistanceConstraint Class Reference

`#include <distance_constraint.h>`

Inheritance diagram for DistanceConstraint



Collaboration diagram for DistanceConstraint:



Public Methods

- **DistanceConstraint** (**ConstraintManager** &cm)
- **DistanceConstraint** (**ConstraintManager** &cm, int atom_i, int atom_j, double distance, double var)
- **~DistanceConstraint** ()
- **jVector dStress** (int k, const **jVector** &pos) const
- int **Compare** (const DistanceConstraint &dc) const
- **MAKECOMPARISONS_WITH_COMPARE** (DistanceConstraint)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const

Static Public Attributes

- **DynString Identifier** = "atomdis:"

Protected Methods

- double **computeStress** () const
- double **computeStress** (int k, const **jVector** &pos) const
- double **computeAbsRes** (void) const
- int **Read** (istream &is)
- int **ReadBinary** (istream &is)

  Same scenerio as **Read***()* (p. 157) *only binary version.*

- void **PreSetup** ()

Protected Attributes

- int **envelopeID**
- double **measured**
- double **variance**
- double **stdev**

4.21.1 Detailed Description

//

distance_constraint.h

This is the first constraint type for GRAPE. It constrains the distance between two atoms.

//

Definition at line 20 of file distance_constraint.h.

4.21.2 Constructor & Destructor Documentation

4.21.2.1 DistanceConstraint::DistanceConstraint (ConstraintManager & *cm*)

Definition at line 3 of file distance_constraint.C.

```
00004   : Constraint(cm,IntVec(2,-1))
00005   {
00006     // this is an unititialized distance constraint.  The two atoms
00007     // and the mesured and variance values must be added.
00008
00009     measured = 0.0;
00010     variance = INF;
00011
```

4.21.2.2 DistanceConstraint::DistanceConstraint (ConstraintManager & *cm*, int *i*, int *j*, double *distance*, double *var*)

Takes two indices of the atoms, the measured distance between them and the variance associated with this measurement. A molecule, M, is also given to serve as a "template" structure from which the object determines the number (and types) of atoms in the molecule of interest.

Definition at line 13 of file distance_constraint.C.

```
00015   : Constraint(cm, cat( IntVec(1,((i<j) ? i : j)),
00016                         IntVec(1,((i<j) ? j : i))))
00017 {
00018   measured = distance;
00019   variance = var;
00020   if (variance<gwNumerics::MinDistVariance)
00021     {gwNumerics::MinDistVariance = variance;}
00022
```

4.21.2.3 DistanceConstraint::~DistanceConstraint () [inline]

Definition at line 36 of file distance_constraint.h.

```
00036                       {}
00037
```

4.21.3 Member Function Documentation

4.21.3.1 int DistanceConstraint::Compare (const DistanceConstraint & *dc*) const [inline]

Definition at line 84 of file distance_constraint.h.

```
00085 {
00086   if(supportA[0] == dc.supportA[0] && supportA[1] == dc.supportA[1])
00087     return 0;
00088   if(supportA[0]<dc.supportA[0])
00089     return -1;
00090   if(supportA[0]>dc.supportA[0])
00091     return 1;
00092   if(supportA[1]<dc.supportA[1])
00093     return -1;
00094   return 1;
00095
```

4.21.3.2 DistanceConstraint::MAKECOMPARISONS_WITH_COMPARE (DistanceConstraint)

4.21.3.3 int DistanceConstraint::Output (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 235 of file distance_constraint.C.

```
00236 {
00237   os << supportA[0] << ' ' << supportA[1] << ' '
00238      << measured << ' ' << variance;
00239   return 2;
00240
```

4.21.3.4 int DistanceConstraint::OutputBinary (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 250 of file distance_constraint.C.

```
00251 {
00252   os <= supportA <= measured <= variance;
00253   return 2;
00254
```

4.21.3.5 void DistanceConstraint::PreSetup (void) [protected, virtual]

The setup function is called if the uninialized constraint method is used to create the constraint to allow different data representations within the constraint. Without this, only data read in from the input file will be available to the member functions. For simple constraints, this is really not very useful. However for multi-atom constraints that need to be a little 'smarter' about data representations, the Setup function will be helpful.

Reimplemented from **Constraint** (p. 85).

Definition at line 25 of file distance_constraint.C.

```
00026 {
00027   //  CString = Identifier;
00028   stdev = sqrt(variance);
00029
```

4.21.3.6 int DistanceConstraint::Read (istream & *is*) [protected, virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 157) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 157) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented from **Constraint** (p. 85).

Definition at line 242 of file distance_constraint.C.

```
00243 {
00244   is >> supportA[0] >> supportA[1] >> measured >> variance;
00245   if (variance<gwNumerics::MinDistVariance)
00246     {gwNumerics::MinDistVariance = variance;}
00247
```

4.21.3.7 int DistanceConstraint::ReadBinary (istream & *is*) [protected, virtual]

Same scenerio as **Read**() (p. 157) only binary version.

Reimplemented from **Constraint** (p. 85).

Definition at line 256 of file distance_constraint.C.

```
00257 {
00258   is >= supportA[0] >= supportA[1] >= measured >= variance;
00259   if (variance<gwNumerics::MinDistVariance)
00260     {gwNumerics::MinDistVariance = variance;}
00261
```

4.21.3.8 double DistanceConstraint::computeAbsRes (void) const [protected, virtual]

returns abs residual as a Z score, if supported, constraint overwrites

Reimplemented from **Constraint** (p. 89).

Definition at line 227 of file distance_constraint.C.

```
00228 {
00229   int i=supportA[0], j=supportA[1];
00230   double calc = euclid(getMolecule()(i),getMolecule()(j));
00231   return fabs((calc-measured)/stdev);
00232
```

4.21.3.9 double DistanceConstraint::computeStress (int *k*, const jVector & *pos_k*) const [protected, virtual]

Returns the stress on this constraint after "pretending" the atom with index a' moved to position pos_a'. This also must be overwritten by all constraints that inherit from this class

Reimplemented from **Constraint** (p. 89).

Definition at line 56 of file distance_constraint.C.

```
00057 {
00058   double calc;  // calculated (current model) distance between i and j.
00059   int i=supportA[0], j=supportA[1];
00060
00061   // If the atom that has moved does not include either of the two
00062   // atoms this constraint is defined over, then just default to reporting
00063   // how stressed we are about our two atoms.
00064   //
00065   if(k != i && k != j){
00066     cerr << "*** Called computeStress("<<k<<","<<pos_k<<") on dist constraint over atoms " ;
00067     cerr << i << " and " << j << "***" << endl;
00068     DIE("");
00069     return computeStress();
00070   }
00071   // Figure out which atom has moved and report our new satisfaction:
00072   //
00073   if(k == i)
00074       calc = euclid(pos_k, getMolecule()(j));
00075   else
00076       calc = euclid(getMolecule()(i), pos_k);
00077
00078   // Our new satisfaction is the difference between the measured and the
00079   // predicted distance:
00080   //
00081 //    cerr << "ORDER cmputstr-pos: " << Label() << ";calc = " << setprecision(50) << calc << endl;
00082
00083   double R = (calc-measured)/stdev;
00084 //   if (R>DistanceConstraint::MaxAbsResidual)
00085 //     {DistanceConstraint::MaxAbsResidual = R;}
00086
```

4.21.3.10 double DistanceConstraint::computeStress () const [protected, virtual]

One of the only functions that *must* be overwritten by subclasses. Returns a value reflecting how dissatisfied this constraint is with the current molecule residing in CM'.

Reimplemented from **Constraint** (p. 89).

Definition at line 36 of file distance_constraint.C.

Referenced by computeStress().

```
00037 {
00038   int i=supportA[0], j=supportA[1];
00039
00040   // Find the calculated distance in the model
00041   double calc = euclid(getMolecule()(i),getMolecule()(j));
00042 //   cerr << "ORDER cmputstr: " << Label() << ";calc = " << setprecision(50) << calc << endl;
00043 //   cerr << "ORDER cmputstr: " << Label() << ";meas = " << setprecision(50) << measured << endl;
00044   double R = (calc-measured)/stdev;
00045 //   cerr << "ORDER cmputstr: " << Label() << ";R' = " << setprecision(50) << R << endl;
00046   //  R = R/stdev;
00047 //   if (R>DistanceConstraint::MaxAbsResidual)
```

```
00048 //     {DistanceConstraint::MaxAbsResidual = R;}
00049
```

4.21.3.11 jVector DistanceConstraint::dStress (int *k*, const jVector & *pos_k*) const [virtual]

asking constraint number k for its 12 element dStress vector we pass in pos_a vector – corresponding to the atom position if the constraint is an atom, and corresponding to the position of the atom k in question if the call is to a non-atom constraint. EG for a distance constraint atom 12 needs the dStress from the d-constraint, so it will call dStress on the d-constraint(12, atom_pos) so the d_constr can answer the question if atom 12 were in atom_pos, I would say my dStress is blah.

Reimplemented from **Constraint** (p. 89).

Definition at line 171 of file distance_constraint.C.

```
00172 {
00173   // A HACK and a TEMPORARY FIX to an inherent logic problems in this
00174   // set of functions.  come see me w/ questions -jd
00175
00176    int i=supportA[0], j=supportA[1];
00177
00178   if(k != i && k != j)
00179     {
00180       cerr << "Fatal error: distance_constraint.C, dStress("<< k \
00181            <<", pos) not legal on atom " << k << endl;
00182       exit(1);
00183     }
00184
00185   // Figure out which atom has moved and report our new satisfaction:
00186   //
00187   double calc;
00188   if(k == i)
00189       calc = euclid(pos_k, getMolecule()(j));
00190   else
00191       calc = euclid(getMolecule()(i), pos_k);
00192
00193 //   cerr << "ORDER dstress: " << Label() << ";calc = " << setprecision(50) << calc << endl;
00194
00195   jVector dist(3,0.0);
00196   //  jVector return_vect(12,0.0);
00197   jVector return_vect(3,0.0);
00198   double R = (calc-measured)/stdev;
00199
00200   //  cerr << "R=" << R << endl;
00201
00202   if(k==i)
00203     dist = ((pos_k - getMolecule()(j))/calc/stdev);
00204   else if(k==j)
00205     dist = ((pos_k - getMolecule()(i))/calc/stdev);
00206   else {
00207     DIE("Error in distance dstress call.");
00208     return return_vect;
00209   }
00210
```

4.21.4 Member Data Documentation

4.21.4.1 DynString DistanceConstraint::Identifier = "atomdis:" [static]

Reimplemented from **Constraint** (p. 95).

Definition at line 100 of file globals.C.

4.21.4.2 int DistanceConstraint::envelopeID [protected]

Definition at line 55 of file distance_constraint.h.

4.21.4.3 double DistanceConstraint::measured [protected]

Definition at line 71 of file distance_constraint.h.

4.21.4.4 double DistanceConstraint::stdev [protected]

Definition at line 73 of file distance_constraint.h.

4.21.4.5 double DistanceConstraint::variance [protected]

Definition at line 72 of file distance_constraint.h.

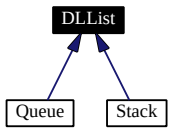
The documentation for this class was generated from the following files:

- **distance_constraint.h**
- **distance_constraint.C**
- **globals.C**

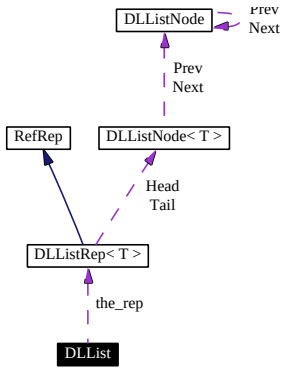
4.22 DLList Class Template Reference

`#include <dllist.h>`

Inheritance diagram for DLList



Collaboration diagram for DLList:



Public Methods

- **DLList** ()
- **DLList** (const DLList< T > &dll)
- **~DLList** ()
- int **IsEmpty** () const
- **operator int** () const
- void **Clear** ()
- int **Number** () const
- bool **AddToFront** (const T &data)
- bool **AddToBack** (const T &data)
- bool **AddToFront** (const DLList< T > &dll)
- bool **AddToBack** (const DLList< T > &dll)
- bool **Add** (const T &data)
- bool **Add** (const DLList< T > &dll)
- const DLList<T>& **operator+=** (const DLList< T > &dll)
- T **RemoveFromFront** ()
- T **RemoveFromBack** ()
- T **Remove** ()
- const T& **Get** (int i) const
- T& **Get** (int i)
- const T& **PeekFront** () const
- T& **PeekFront** ()
- const T& **PeekBack** () const

- T& **PeekBack** ()
- T **RemoveLocation** (int i)
- T **Remove** (const T &data)
- int **operator==** (const DLList< T > &dll) const
- int **operator!=** (const DLList< T > &dll) const
- T& **Find** (const T &data)
- const T& **Find** (const T &data) const
- int **FindLocation** (const T &data) const
- int **In** (const T &data) const
- ostream& **Print** (ostream &os) const
- DLList<T> **UniqueCopy** () const
- const DLList<T>& **operator=** (const DLList< T > &dll)
- DLList<T>::**Iterator GetIterator** ()
- DLList<T>::**ConstIterator GetConstIterator** () const

Private Attributes

- **DLListRep**<T>* **the_rep**

Friends

- class **Iterator**
- class **ConstIterator**
- ostream& **operator** (ostream &os, const DLList< T > &dll)
- istream& **InputBinary** (istream &is, DLList< T > &dll)
- istream& **operator>=** (istream &is, DLList< T > &dll)
- ostream& **OutputBinary** (ostream &os, const DLList< T > &dll)
- ostream& **operator** (ostream &os, const DLList< T > &dll)

template<class T> class DLList

4.22.1 Constructor & Destructor Documentation

4.22.1.1 template<class T> DLList< T >::DLList<T> () [inline]

Definition at line 459 of file dllist.inc.

```
00460   { the_rep = new DLListRep<T>(); }
00461
```

4.22.1.2 template<class T> DLList< T >::DLList<T> (const DLList< T > & *dll*) [inline]

Definition at line 463 of file dllist.inc.

```
00464   {
00465     the_rep = dll.the_rep;
00466     if(the_rep) the_rep->Reference();
00467
```

4.22.1.3 template<class T> DLList< T >::~DLList<T> () [inline]

Definition at line 470 of file dllist.inc.

```
00471   {
00472     if(the_rep)
00473         the_rep->Dereference();
00474     the_rep = 0;
00475
```

4.22.2 Member Function Documentation

4.22.2.1 template<class T> bool DLList< T >::Add (const DLList< T > & *dll*) [inline]

Definition at line 162 of file dllist.inc.

```
00163   { return AddToBack(dll); }
00164
```

4.22.2.2 template<class T> bool DLList< T >::Add (const T & *x*) [inline]

Reimplemented in **Queue** (p. 401).

Definition at line 158 of file dllist.inc.

Referenced by Queue::operator+=(), and Molecule::readFasta().

```
00159   { return AddToBack(data); }
00160
```

4.22.2.3 template<class T> bool DLList< T >::AddToBack (const DLList< T > & *dll*) [inline]

Definition at line 127 of file dllist.inc.

```
00128 {
00129   if(!dll.the_rep->Head) return false;
00130   if(IsEmpty()) { *this = dll.UniqueCopy(); return false; }
00131
00132   DLListNode<T> * Hop, * New, * ExtPoint, * Previous;
00133
00134   Hop      = dll.the_rep->Head;
00135   New      = new DLListNode<T>(Hop->Data);
00136   ExtPoint = New;
00137
00138   Hop = Hop->Next;
00139   while(Hop)
00140     {
00141       Previous = New;
00142       New = new DLListNode<T>(Hop->Data);
00143       New->Prev = Previous;
00144       Previous->Next = New;
00145       Hop = Hop->Next;
00146     }
00147
```

4.22.2.4 template<class T> bool DLList< T >::AddToBack (const T & *data*) [inline]

Definition at line 68 of file dllist.inc.

Referenced by Queue::Add(), Add(), Molecule::DynRead(), InputBinary(), Segments::QSort-Reduce(), sdGraph::TravLists(), UniqueCopy(), Process::addChild(), jVector::dynread(), Int-Vec::dynread(), ConstraintManager::getAssocAtoms(), ConstraintManager::getParents(), main(), and operator+=().

```
00069 {
00070   // Create a new node.
00071   //
00072   DLListNode<T> * New = new DLListNode<T>(data);
00073
00074   // Make the node the new tail of the list.
00075   //
00076   if(IsEmpty())
00077     {
00078       New->Next = 0;
00079       New->Prev = 0;
00080       the_rep->Head = New;
00081       the_rep->Tail = New;
00082     }
00083   else
00084     {
00085       the_rep->Tail->Next = New;
00086       New->Prev  = the_rep->Tail;
00087       New->Next  = 0;
00088       the_rep->Tail = New;
00089     }
00090
```

4.22.2.5 template<class T> bool DLList< T >::AddToFront (const DLList< T > & *dll*) [inline]

Definition at line 96 of file dllist.inc.

```
00097 {
00098   if(!dll.the_rep->Head) return false;
00099   if(IsEmpty()) { *this = dll.UniqueCopy(); return false; }
00100
00101   DLListNode<T> * Hop, * New, * NewHead, * Previous;
00102
00103   Hop     = dll.the_rep->Head;
00104   New     = new DLListNode<T>(Hop->Data);
00105   NewHead = New;
00106
00107   Hop = Hop->Next;
00108   while(Hop)
00109     {
00110       Previous = New;
00111       New = new DLListNode<T>(Hop->Data);
00112       New->Prev = Previous;
00113       Previous->Next = New;
00114       Hop = Hop->Next;
00115     }
00116
```

4.22.2.6 template<class T> bool DLList< T >::AddToFront (const T & *data*) [inline]

Definition at line 41 of file dllist.inc.

Referenced by Segments::Add(), Stack::Push(), Process::addChild(), Grid::approx_neighbor_i(), Grid::approx_neighbor_p(), Grid::neighbor_d(), Grid::neighbor_i(), and Grid::neighbor_p().

```
00042 {
00043   // Create a new node.
00044   //
00045   DLListNode<T> * New = new DLListNode<T>(data);
00046
00047   // Make the node the new head of the list.
00048   //
00049   if(IsEmpty())
00050     {
00051       New->Next = 0;
00052       New->Prev = 0;
00053       the_rep->Head = New;
00054       the_rep->Tail = New;
00055     }
00056   else
00057     {
00058       the_rep->Head->Prev = New;
00059       New->Next  = the_rep->Head;
00060       New->Prev  = 0;
00061       the_rep->Head = New;
00062     }
00063
```

4.22.2.7 template<class T> void DLList< T >::Clear () [inline]

Definition at line 33 of file dllist.inc.

Referenced by Segments::Clear(), InputBinary(), Segments::QSortReduce(), IdGen::Reset(), bf_iterator::reset(), and df_iterator::reset().

```
00034   { delete the_rep->Head; the_rep->Head=0; the_rep->Tail=0; the_rep->M=0; }
00035
```

4.22.2.8 template<class T> const T & DLList< T >::Find (const T & *data*) const [inline]

Definition at line 360 of file dllist.inc.

```
00361   { return Get( FindLocation(data) ); }
00362
```

4.22.2.9 template<class T> T & DLList< T >::Find (const T & *data*) [inline]

Definition at line 356 of file dllist.inc.

```
00357   { return Get( FindLocation(data) ); }
00358
```

4.22.2.10 template<class T> int DLList< T >::FindLocation (const T & *data*) const [inline]

Definition at line 344 of file dllist.inc.

Referenced by Find(), In(), and Remove().

```
00345 {
00346   DLListNode<T> * Hop = the_rep->Head;
00347   for(int i=0; Hop; ++i)
00348     {
00349       if(Hop->Data == data) return i;
00350       Hop = Hop->Next;
00351     }
00352
```

4.22.2.11 template<class T> T & DLList< T >::Get (int *i*) [inline]

Definition at line 252 of file dllist.inc.

```
00253 {
00254   if((i>=the_rep->M) || (i<0)) i=0;
00255
00256   DLListNode<T> * Hop;
00257   if(i<the_rep->M/2)
00258     {
00259       Hop = the_rep->Head;
00260       for(int j=0; j<i; ++j)
00261           Hop = Hop->Next;
00262     }
00263   else
00264     {
00265       Hop = the_rep->Tail;
00266       for(int j=the_rep->M-1; j>i; --j)
00267           Hop = Hop->Prev;
00268     }
00269
```

4.22.2.12 template<class T> const T & DLList< T >::Get (int *i*) const [inline]

Definition at line 231 of file dllist.inc.

Referenced by Find(), Segments::GetSeg(), Stack::Peek(), Segments::QSortReduceViol(), and Process::runChild().

```
00232 {
00233   if((i>=the_rep->M) || (i<0)) i=0;
00234
00235   DLListNode<T> * Hop;
00236   if(i<the_rep->M/2)
00237     {
00238       Hop = the_rep->Head;
00239       for(int j=0; j<i; ++j)
00240           Hop = Hop->Next;
00241     }
00242   else
00243     {
00244       Hop = the_rep->Tail;
00245       for(int j=the_rep->M-1; j>i; --j)
00246           Hop = Hop->Prev;
```

```
00247     }
00248
```

4.22.2.13 template<class T> DLList< T >::ConstIterator DLList<T>::GetConstIterator<T> () const [inline]

Definition at line 170 of file dllist.h.

Referenced by gwNumerics::addVDWSegments(), ConstraintManager::getAssocAtoms(), and Molecule::readFasta().

```
00171                 { return DLList<T>::ConstIterator(*this); }
00172
```

4.22.2.14 template<class T> DLList< T >::Iterator DLList<T>::GetIterator<T> () [inline]

Definition at line 167 of file dllist.h.

Referenced by OutputBinary(), and path_iterator::path_iterator().

```
00168                 { return DLList<T>::Iterator(*this); }
00169
```

4.22.2.15 template<class T> int DLList< T >::In (const T & *data*) const [inline]

Definition at line 364 of file dllist.inc.

```
00365   { return (FindLocation(data) != -1); }
00366
```

4.22.2.16 template<class T> int DLList< T >::IsEmpty () const [inline]

Definition at line 29 of file dllist.inc.

Referenced by AddToBack(), AddToFront(), IdGen::Get(), IdGen::Peek(), main(), operator int(), bf_iterator::operator++(), and df_iterator::operator++().

```
00030   { return (the_rep->Head ? 0 : 1); }
00031
```

4.22.2.17 template<class T> int DLList< T >::Number () const [inline]

Definition at line 37 of file dllist.inc.

Referenced by Molecule::DynRead(), Segments::GetNum(), Segments::Number(), IdGen::NumberFree(), OutputBinary(), Segments::QSortReduce(), Segments::QSortReduceViol(), Process::assertOrder(), jVector::dynread(), IntVec::dynread(), Process::numQueued(), Molecule::readFasta(), and Process::runChild().

```
00038   { return the_rep->M; }
00039
```

4.22.2.18 template<class T> T & DLList< T >::PeekBack () [inline]

Definition at line 285 of file dllist.inc.

```
00286   { return the_rep->Tail->Data; }
00287
```

4.22.2.19 template<class T> const T & DLList< T >::PeekBack () const [inline]

Definition at line 281 of file dllist.inc.
Referenced by Process::addChild().

```
00282   { return the_rep->Tail->Data; }
00283
```

4.22.2.20 template<class T> T & DLList< T >::PeekFront () [inline]

Definition at line 277 of file dllist.inc.

```
00278   { return the_rep->Head->Data; }
00279
```

4.22.2.21 template<class T> const T & DLList< T >::PeekFront () const [inline]

Definition at line 273 of file dllist.inc.
Referenced by Queue::Peek(), and Process::addChild().

```
00274   { return the_rep->Head->Data; }
00275
```

4.22.2.22 template<class T> ostream & DLList< T >::Print (ostream & *os*) const [inline]

Definition at line 372 of file dllist.inc.

```
00373 {
00374   DLListNode<T> * Hop = the_rep->Head;
00375   while(Hop)
00376   {
00377     os << Hop->Data << " -> ";
00378     Hop = Hop->Next;
00379   }
00380
```

4.22.2.23 template<class T> T DLList< T >::Remove (const T & *data*) [inline]

Definition at line 368 of file dllist.inc.

```
00369   { return RemoveLocation( FindLocation(data) ); }
00370
```

4.22.2.24 template<class T> T DLList< T >::Remove () [inline]

Reimplemented in **Queue** (p. 401).

Definition at line 227 of file dllist.inc.

Referenced by Molecule::readFasta().

```
00228   { return RemoveFromFront(); }
00229
```

4.22.2.25 template<class T> T DLList< T >::RemoveFromBack () [inline]

Definition at line 202 of file dllist.inc.

Referenced by RemoveLocation(), Process::assertOrder(), Process::cleanQueue(), main(), Process::runChildren(), Process::runChildrenRepeatFirst(), and Process::runNextChild().

```
00203 {
00204   T DeletedData;
00205
00206   //  if(!the_rep->Tail) return DeletedData;  // gives warning
00207   if(the_rep->Tail){
00208
00209     DLListNode<T> * tmp = the_rep->Tail;
00210     the_rep->Tail = the_rep->Tail->Prev;
00211     if(the_rep->Tail) the_rep->Tail->Next = 0;
00212     else        the_rep->Head = 0;
00213
00214     DeletedData = tmp->Data;
00215     tmp->Prev = 0;
00216     delete tmp;
00217     --the_rep->M;
00218     return DeletedData;
00219   } else {
00220     cerr << "called RemoveFromBack from empty dllist. __FILE__, __LINE__" << endl;
00221     raise (SIGQUIT);
00222     return the_rep->Head->Data; // GD useless line -- warning removal only
00223   }
00224
```

4.22.2.26 template<class T> T DLList< T >::RemoveFromFront () [inline]

Definition at line 176 of file dllist.inc.

Referenced by Molecule::DynRead(), Stack::Pop(), Segments::Pop(), Segments::QSortReduce(), Queue::Remove(), Remove(), RemoveLocation(), jVector::dynread(), IntVec::dynread(), and main().

```
00177 {
00178   T DeletedData;
00179
00180   //  if(!the_rep->Head) return DeletedData; // gives warning for non-init
00181
00182   if(the_rep->Head){
00183
00184     DLListNode<T> * tmp = the_rep->Head;
00185     the_rep->Head = the_rep->Head->Next;
00186     if(the_rep->Head) the_rep->Head->Prev = 0;
00187     else           the_rep->Tail = 0;
```

```
00188
00189     DeletedData = tmp->Data;
00190     tmp->Next = 0;
00191     delete tmp;
00192     --the_rep->M;
00193     return DeletedData;
00194   } else {
00195     cerr << "called RemoveFromFront from empty dllist. __FILE__, __LINE__" << endl;
00196     raise (SIGQUIT);
00197     return the_rep->Head->Data; // GD useless line -- warning removal only
00198   }
00199
```

4.22.2.27 template<class T> T DLList< T >::RemoveLocation (int *i*) [inline]

Definition at line 290 of file dllist.inc.

Referenced by Remove().

```
00291 {
00292   if(i<=0) { return RemoveFromFront(); }
00293   if(i>=the_rep->M-1) { return RemoveFromBack(); }
00294
00295   DLListNode<T> * Hop;
00296   if(i<the_rep->M/2)
00297     {
00298       Hop = the_rep->Head;
00299       for(int j=0; j<i; ++j)
00300           Hop = Hop->Next;
00301     }
00302   else
00303     {
00304       Hop = the_rep->Tail;
00305       for(int j=the_rep->M-1; j>i; --j)
00306           Hop = Hop->Prev;
00307     }
00308
```

4.22.2.28 template<class T> DLList< T > DLList< T >::UniqueCopy () const [inline]

Definition at line 446 of file dllist.inc.

Referenced by AddToBack(), and AddToFront().

```
00447   {
00448     DLList<T> dll;
00449     DLListNode<T> * Hop = the_rep->Head;
00450     while(Hop)
00451       {
00452         dll.AddToBack(Hop->Data);
00453         Hop = Hop->Next;
00454       }
00455
```

4.22.2.29 template<class T> DLList<T>::operator int () const [inline]

Definition at line 94 of file dllist.h.

```
00095                   { return !IsEmpty(); }
00096
```

4.22.2.30 template<class T> int DLList< T >::operator!= (const DLList< T > & *dll*) const [inline]

Definition at line 340 of file dllist.inc.

```
00341   { return !( this->operator==(dll) ); }
00342
```

4.22.2.31 template<class T> const DLList< T > & DLList< T >::operator+= (const DLList< T > & *dll*) [inline]

Definition at line 166 of file dllist.inc.

```
00167   { AddToBack(dll); return *this; }
00168
```

4.22.2.32 template<class T> const DLList< T > & DLList< T >::operator= (const DLList< T > & *dll*) [inline]

Definition at line 435 of file dllist.inc.

```
00436   {
00437     if(the_rep == dll.the_rep) return *this;
00438
00439     if(the_rep) the_rep->Dereference();
00440     the_rep = dll.the_rep;
00441     if(the_rep) the_rep->Reference();
00442     return *this;
00443
```

4.22.2.33 template<class T> int DLList< T >::operator== (const DLList< T > & *dll*) const [inline]

Definition at line 323 of file dllist.inc.

Referenced by operator!=().

```
00324   {
00325     if(the_rep->M != dll.the_rep->M) return 0;
00326     if(the_rep->M == 0) return 1;
00327
00328     DLListNode<T> * Hop1 = the_rep->Head;
00329     DLListNode<T> * Hop2 = dll.the_rep->Head;
00330     while(Hop1)
00331       {
00332         if(Hop1->Data != Hop2->Data) return 0;
00333         Hop1 = Hop1->Next;
00334         Hop2 = Hop2->Next;
00335       }
00336
```

4.22.3 Friends And Related Function Documentation

4.22.3.1 template<class T> class ConstIterator [friend]

Definition at line 82 of file dllist.h.

Referenced by GetConstIterator().

4.22.3.2 template<class T> istream& InputBinary (istream & *is*, DLList< T > & *dll*) [friend]

Definition at line 389 of file dllist.inc.

```
00390   {
00391     // Delete everything:
00392     //
00393     dll.Clear();
00394 
00395     // Read in the number of elements:
00396     //
00397     int numel;
00398     if(InputBinary(is,numel).bad())
00399         return is;
00400 
00401     T stuff;
00402     for(int i=0; i<numel; ++i)
00403       {
00404         if(InputBinary(is,stuff).bad())
00405             return is;
00406         dll.AddToBack(stuff);
00407       }
00408 
```

4.22.3.3 template<class T> class Iterator [friend]

Definition at line 81 of file dllist.h.

Referenced by GetIterator().

4.22.3.4 template<class T> ostream& OutputBinary (ostream & *os*, const DLList< T > & *dll*) [friend]

Definition at line 412 of file dllist.inc.

```
00413   {
00414     // Write the number of elements:
00415     //
00416     if(OutputBinary(os, dll.Number()).bad())
00417         return os;
00418 
00419     for(DLList<T>::Iterator it = dll.GetIterator(); it; ++it)
00420         if(OutputBinary(os, *it).bad())
00421             return os;
00422     return os;
00423 
```

4.22.3.5 template<class T> ostream& operator (ostream & *os*, const DLList< T > & *dll*) [friend]

4.22.3.6 template<class T> ostream& operator (ostream & *os*, const DLList< T > & *dll*) [friend]

4.22.3.7 template<class T> istream& operator>= (istream & *is*, DLList< T > & *dll*) [friend]

Definition at line 427 of file dllist.inc.

```
00428   { return InputBinary(os,dll); }
00429
```

4.22.4 Member Data Documentation

4.22.4.1 template<class T> DLListRep< T > * DLList<T>::the_rep [private]

Definition at line 174 of file dllist.h.

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

- **dllist.h**
- **dllist.inc**

4.23 DLListNode Class Template Reference

`#include <dllist.h>`

Collaboration diagram for DLListNode:

Public Methods

- **DLListNode** ()
- **DLListNode** (const T &data)
- **~DLListNode** ()

Public Attributes

- T **Data**
- DLListNode* **Next**
- DLListNode* **Prev**

Friends

- class **DLList< T >**
- class **DLListRep< T >**

template<class T> class DLListNode

4.23.1 Constructor & Destructor Documentation

4.23.1.1 template<class T> DLListNode<T>::DLListNode<T> () [inline]

Definition at line 36 of file dllist.h.

```
00037                 { Next = 0; Prev = 0; }
00038
```

4.23.1.2 template<class T> DLListNode<T>::DLListNode<T> (const T & *data*) [inline]

Definition at line 38 of file dllist.h.

```
00039                 { Next = 0; Prev = 0; Data = data; }
00040
```

4.23.1.3 template<class T> DLListNode<T>::~DLListNode<T> () [inline]

Definition at line 42 of file dllist.h.

```
00043                 { if(Next) delete Next; }
00044
```

4.23.2 Friends And Related Function Documentation

4.23.2.1 template<class T> class DLList [friend]

Definition at line 31 of file dllist.h.

4.23.2.2 template<class T> class DLListRep [friend]

Definition at line 32 of file dllist.h.

4.23.3 Member Data Documentation

4.23.3.1 template<class T> T DLListNode<T>::Data

Definition at line 46 of file dllist.h.

4.23.3.2 template<class T> DLListNode<T> * DLListNode<T>::Next

Definition at line 48 of file dllist.h.

4.23.3.3 template<class T> DLListNode<T> * DLListNode<T>::Prev

Definition at line 50 of file dllist.h.

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

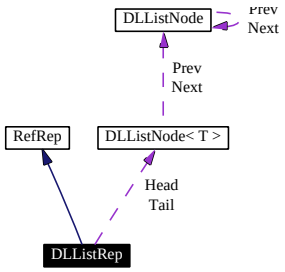
- **dllist.h**

4.24 DLListRep Class Template Reference

`#include <dllist.h>`

Inheritance diagram for DLListRep

Collaboration diagram for DLListRep:



Public Methods

- **DLListRep** ()
- **~DLListRep** ()

Public Attributes

- int **M**
- **DLListNode**<T>* **Head**
- **DLListNode**<T>* **Tail**

Friends

- class **DLList< T >**

template<class T> class DLListRep

4.24.1 Constructor & Destructor Documentation

4.24.1.1 template<class T> DLListRep< T >::DLListRep<T> () [inline]

Definition at line 15 of file dllist.inc.

```
00016   { M=0; Head=0; Tail=0; }
00017
```

4.24.1.2 template<class T> DLListRep< T >::~DLListRep<T> () [inline]

Definition at line 19 of file dllist.inc.

```
00020   { if(Head) delete Head; }
00021
```

4.24.2 Friends And Related Function Documentation

4.24.2.1 template<class T> class DLList [friend]

Definition at line 60 of file dllist.h.

4.24.3 Member Data Documentation

4.24.3.1 template<class T> DLListNode< T > * DLListRep<T>::Head

Definition at line 68 of file dllist.h.

4.24.3.2 template<class T> int DLListRep<T>::M

Definition at line 65 of file dllist.h.

4.24.3.3 template<class T> DLListNode< T > * DLListRep<T>::Tail

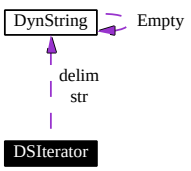
Definition at line 71 of file dllist.h.

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

- **dllist.h**
- **dllist.inc**

4.25 DSIterator Class Reference

Collaboration diagram for DSIterator:



Public Methods

- void **Reset** ()
- void **SetString** (const **DynString** &newstring)
- void **SetDelim** (const **DynString** &newdelims)
- **DynString operator** * ()
- **operator int** ()
- const DSIterator& **operator++** ()
- void **Copy** (const DSIterator &it)
- const DSIterator& **operator=** (const DSIterator &it)
- **DSIterator** ()
- **DSIterator** (const DSIterator &it)
- **DSIterator** (const **DynString** &ds, const **DynString** &delims)
- **DSIterator** (const **DynString** &ds)
- **~DSIterator** ()

Private Methods

- int **DelimMatch** (char c)
- void **Position** ()

Private Attributes

- **DynString str**
- **DynString delim**
- int **b**
- int **e**
- int **reset**

4.25.1 Constructor & Destructor Documentation

4.25.1.1 DSIterator::DSIterator () [inline]

Definition at line 57 of file dsiter.inc.

```
00057                           { b=0; e=0; reset=0; }
00058
```

4.25.1.2 DSIterator::DSIterator (const DSIterator & *it*) [inline]

Definition at line 59 of file dsiter.inc.

```
00060                   { Copy(it); }
00061
```

4.25.1.3 DSIterator::DSIterator (const DynString & *ds*, const DynString & *delims*) [inline]

Definition at line 62 of file dsiter.inc.

```
00063                   {
00064                     str = ds;
00065                     delim = delims;
00066                     Reset();
00067                     reset = 0;
00068
```

4.25.1.4 DSIterator::DSIterator (const DynString & *ds*) [inline]

Definition at line 70 of file dsiter.inc.

```
00071                   {
00072                     str = ds;
00073                     delim = GetDSDelim();
00074                     Reset();
00075                     reset = 0;
00076
```

4.25.1.5 DSIterator::~DSIterator () [inline]

Definition at line 78 of file dsiter.inc.

```
00078                               {}
00079
```

4.25.2 Member Function Documentation

4.25.2.1 void DSIterator::Copy (const DSIterator & *it*) [inline]

Definition at line 45 of file dsiter.inc.
Referenced by DSIterator(), and operator=().

```
00046                   {
00047                     str = it.str;
00048                     delim = it.delim;
00049                     b = it.b;
00050                     e = it.e;
00051                     reset = it.reset;
00052
```

4.25.2.2 int DSIterator::DelimMatch (char *c*) [inline, private]

Definition at line 89 of file dsiter.inc.
Referenced by Position().

```
00090                   {
00091                     for(int i=0; i<delim.Length(); ++i)
00092                         if(delim[i] == c)
00093                             return 1;
00094                     return 0;
00095
```

4.25.2.3 void DSIterator::Position () [inline, private]

Definition at line 97 of file dsiter.inc.
Referenced by Reset(), and operator++().

```
00098                   {
00099                     // Move the beginning to the next non-delimiter:
00100                     for(b=e+1; b<str.Length() && DelimMatch(str[b]); ++b);
00101
00102                     // Move the end to the position of the last non-delimiter
00103                     // after the beginning:
00104                     for(e=b; e<str.Length()-1 && !DelimMatch(str[e+1]); ++e);
00105
```

4.25.2.4 void DSIterator::Reset () [inline]

Definition at line 13 of file dsiter.inc.
Referenced by DSIterator(), operator *(), operator int(), and operator++().

```
00014                   { e = -1; Position(); }
00015
```

4.25.2.5 void DSIterator::SetDelim (const DynString & *newdelims*) [inline]

Definition at line 20 of file dsiter.inc.

```
00021                   { delim = newdelims; }
00022
```

4.25.2.6 void DSIterator::SetString (const DynString & *newstring*) [inline]

Definition at line 16 of file dsiter.inc.

```
00017                   { str = newstring; reset=1; }
00018
```

4.25.2.7 DynString DSIterator::operator * () [inline]

Definition at line 25 of file dsiter.inc.

```
00026                   {
00027                     if(reset) { Reset(); reset=0; }
00028
```

4.25.2.8 DSIterator::operator int () [inline]

Definition at line 31 of file dsiter.inc.

```
00032                   {
00033                     if(reset) { Reset(); reset=0; }
00034
```

4.25.2.9 const DSIterator & DSIterator::operator++ () [inline]

Definition at line 37 of file dsiter.inc.

```
00038                   {
00039                     if(reset) { Reset(); reset=0; }
00040
```

4.25.2.10 const DSIterator & DSIterator::operator= (const DSIterator & *it*) [inline]

Definition at line 54 of file dsiter.inc.

```
00055                   { Copy(it); return *this; }
00056
```

4.25.3 Member Data Documentation

4.25.3.1 int DSIterator::b [private]

Definition at line 85 of file dsiter.inc.

4.25.3.2 DynString DSIterator::delim [private]

Definition at line 83 of file dsiter.inc.

4.25.3.3 int DSIterator::e [private]

Definition at line 85 of file dsiter.inc.

4.25.3.4 int DSIterator::reset [private]

Definition at line 87 of file dsiter.inc.

4.25.3.5 DynString DSIterator::str [private]

Definition at line 81 of file dsiter.inc.

Referenced by operator *().

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

- **dsiter.inc**

4.26 DynString Class Reference

`#include <dynstring.h>`

Collaboration diagram for DynString:

Public Methods

- **DynString** ()
- **DynString** (const char *str)
- **DynString** (const DynString &ds)
- **DynString** (char c)
- **DynString** (int i)
- **DynString** (double d)
- **~DynString** ()
- int **IsBad** () const
- int **DestResizeToFit** (int nchars)
- int **NonDestResizeToFit** (int nchars)
- int **Resize** (int newlength)
- int **Set** (const DynString &ds)
- int **Set** (const char *str, int numchars)
- int **Set** (int i, char c)
- int **Chop** (int n=1)
- void **Clear** ()
- const DynString& **Cat** (char c)
- const DynString& **Cat** (const char *str)
- const DynString& **Cat** (const DynString &ds)
- DynString **Sub** (int i, int j) const
- DynString **operator()** (int i, int j) const
- int **IsSub** (const DynString &substr) const
- int **Replace** (const DynString &oldstr, const DynString &newstr)
- void **ToUpper** ()
- void **ToLower** ()
- double **ToDbl** () const
- int **ToInt** () const
- int **Compare** (const DynString &ds) const
- const char* **GetData** () const
- char* **GetDataCopy** () const
- int **Length** () const
- int **Bytes** () const
- int **Blocks** () const
- const DynString& **Pad** (char c, int n)
- const DynString& **operator+=** (const DynString &ds)
- const char& **operator[]** (int i) const
- char& **operator[]** (int i)
- const DynString& **operator=** (const DynString &ds)
- const DynString& **operator=** (const char *str)

- const DynString& **operator=** (char c)
- const DynString& **operator=** (double number)
- int **GetChar** (istream &is)
- int **Read** (istream &is, bool append=false)
- int **Split** (DynString *&substr) const
- int **Output** (ostream &os) const
- int **Input** (istream &is)
- int **InputBinary** (istream &is)
- int **OutputBinary** (ostream &os) const
- **MAKECOMPARISONS_WITH_COMPARE** (DynString)
- **MAKE_IO_STREAM_OPERATORS** (DynString)
- **MAKE_BIN_STREAM_OPERATORS** (DynString)
- istream& **ReadLine** (istream &is, bool append=false)
- void **EatLine** (istream &is) const

Static Public Attributes

- char* **Bad** = 0
- const int **BlockSize** = 32
- int **Delimiter** [256]
- DynString **Empty** = ""

Private Methods

- void **MakeBad** ()
- int **DestResize** (int nblocks)
- int **NonDestResize** (int nblocks)

Private Attributes

- char* **innards**
- int **N**
- int **L**

Friends

- void **SetDSDelim** (const DynString &delim)
- void **AddDSDelim** (char delim)
- void **RemDSDelim** (char delim)
- DynString **GetDSDelim** ()
- DynString **IntToDS** (int i)
- DynString **Reproduce** (char c, int n)
- DynString **operator+** (DynString x, const DynString &y)
- int **DSprintf** (DynString &ds, const char *cntl,...)

4.26.1 Constructor & Destructor Documentation

4.26.1.1 DynString::DynString ()

Definition at line 186 of file dynstring.C.
Referenced by EatLine(), and ReadLine().

```
00187   {
00188     innards = 0; N = 0; L = 0;
00189     if(!DestResize(1)) MakeBad();
00190
```

4.26.1.2 DynString::DynString (const char * *str*)

Definition at line 192 of file dynstring.C.

```
00193   {
00194     innards = 0; N = 0; L = 0;
00195     Set(str,strlen(str));
00196
```

4.26.1.3 DynString::DynString (const DynString & *ds*)

Definition at line 198 of file dynstring.C.

```
00199   {
00200     innards = 0; N = 0; L = 0;
00201     Set(ds);
00202
```

4.26.1.4 DynString::DynString (char *c*)

Definition at line 204 of file dynstring.C.

```
00205   {
00206     innards = 0; N = 0; L = 0;
00207     char * str = new char[2];
00208     str[0] = c; str[1] = 0;
00209     Set(str,1);
00210     delete str;
00211
```

4.26.1.5 DynString::DynString (int *i*)

Definition at line 213 of file dynstring.C.

```
00214   {
00215     innards = 0; N = 0; L = 0;
00216     char * str = new char[100];
00217     sprintf(str, "%i", i);
00218     Set(str,strlen(str));
00219     delete str;
00220
```

4.26.1.6 DynString::DynString (double *d*)

Definition at line 222 of file dynstring.C.

```
00223   {
00224     innards = 0; N = 0; L = 0;
00225     char * str = new char[100];
00226     sprintf(str, "%g", d);
00227     Set(str,strlen(str));
00228     delete str;
00229
```

4.26.1.7 DynString::~DynString ()

Definition at line 231 of file dynstring.C.

```
00232 {
00233   if(!IsBad()) delete[] innards;
00234
```

4.26.2 Member Function Documentation

4.26.2.1 int DynString::Blocks () const [inline]

Definition at line 141 of file dynstring.h.

```
00142   { return N; }
00143
```

4.26.2.2 int DynString::Bytes () const [inline]

Definition at line 138 of file dynstring.h.

```
00139   { return N*BlockSize; }
00140
```

4.26.2.3 const DynString & DynString::Cat (const DynString & *ds*)

Definition at line 106 of file dynstring.C.

```
00107   {
00108     if(IsBad() || ds.IsBad()) return *this;
00109     int newlen = L+ds.L;
00110     if(!NonDestResizeToFit(newlen)) return *this;
00111     strncat(innards, ds.innards, ds.L);
00112     L += ds.L;
00113     innards[L] = '\0';
00114     return *this;
00115
```

4.26.2.4 const DynString& DynString::Cat (const char * *str*)

4.26.2.5 const DynString & DynString::Cat (char *c*)

Definition at line 97 of file dynstring.C.
Referenced by GetChar(), Pad(), Read(), Split(), and operator+=().

```
00098   {
00099     if(IsBad()) return *this;
00100     if(!NonDestResizeToFit(L+1)) return *this;
00101     innards[L++] = c;
00102     innards[L]   = '\0';
00103     return *this;
00104
```

4.26.2.6 int DynString::Chop (int $n = 1$)

Definition at line 175 of file dynstring.C.

```
00176   {
00177     if(IsBad() || n<0) return 0;
00178     if(n==0) return L;
00179     n = (n<=L) ? n : L;
00180     NonDestResizeToFit(L-n);
00181     L = L-n;
00182     innards[L] = '\0';
00183     return L;
00184
```

4.26.2.7 void DynString::Clear () [inline]

Definition at line 83 of file dynstring.h.
Referenced by Read(), Residue::Residue(), and Molecule::readFasta().

```
00084   {
00085     if(!DestResize(1)) MakeBad();
00086     else { L=0; innards[0] = '\0'; }
00087
```

4.26.2.8 int DynString::Compare (const DynString & *ds*) const [inline]

Definition at line 122 of file dynstring.h.

```
00123   { return strcmp(innards,ds.innards); }
00124
```

4.26.2.9 int DynString::DestResize (int *nblocks*) [private]

Definition at line 69 of file dynstring.C.
Referenced by Clear(), DestResizeToFit(), DynString(), and Set().

```
00070   {
00071     if(N == nblocks) return 1;
00072     if(nblocks<1) return 0;
00073     char * tmp = innards;
00074     if(!(innards = new char[nblocks*BlockSize]))
00075       { innards = tmp; return 0; }
00076
```

4.26.2.10 int DynString::DestResizeToFit (int *nchars*) [inline]

Definition at line 45 of file dynstring.h.

Referenced by RanWord::Get(), GetDSDelim(), InputBinary(), Replace(), Reproduce(), Set(), Sub(), and operator=().

```
00046   { return DestResize((nchars/BlockSize)+1); }
00047
```

4.26.2.11 void DynString::EatLine (istream & *is*) const

Definition at line 770 of file dynstring.C.

Referenced by main().

```
00771   { DynString().ReadLine(is); }
00772
```

4.26.2.12 int DynString::GetChar (istream & *is*)

Definition at line 451 of file dynstring.C.

Referenced by Molecule::readFasta().

```
00452   {
00453     char c;
00454     is.get(c);
00455     if(is.eof() || is.bad()) return 0;
00456     Cat(c);
00457     return 1;
00458
```

4.26.2.13 const char * DynString::GetData () const [inline]

Definition at line 128 of file dynstring.h.

Referenced by main().

```
00129   { return innards; }
00130
```

4.26.2.14 char * DynString::GetDataCopy () const

Definition at line 412 of file dynstring.C.

```
00413   {
00414     if(IsBad()) return Bad;
00415     char * s = new char[L+1];
00416     strncpy(s,innards,L);
00417     s[L]='\0';
00418     return s;
00419
```

4.26.2.15 int DynString::Input (istream & *is*)

Definition at line 742 of file dynstring.C.

```
00743   { return Read(is); }
00744
```

4.26.2.16 int DynString::InputBinary (istream & *is*)

Definition at line 421 of file dynstring.C.

```
00422   {
00423     // Delete everything
00424     //
00425     delete[] innards; innards = 0;
00426     N = 0;
00427     L = 0;
00428
00429     int num_chars=0;
00430     is >= num_chars;
00431
00432     if(!DestResizeToFit(num_chars))
00433         { cerr << "DynString::InputBinary\n"; raise(SIGQUIT); }
00434
```

4.26.2.17 int DynString::IsBad () const [inline]

Definition at line 41 of file dynstring.h.

Referenced by Cat(), Chop(), GetDataCopy(), IsSub(), Output(), Set(), Sub(), and ~DynString().

```
00042   { return (innards == Bad); }
00043
```

4.26.2.18 int DynString::IsSub (const DynString & *substr*) const

Definition at line 266 of file dynstring.C.

```
00267   {
00268     if(IsBad() || substr.IsBad()) return 0;
00269
00270     int match;
00271     for(int i=0; i<L-substr.L+1; ++i)
00272       {
00273         match = 1;
00274         for(int j=0; j<substr.L; ++j)
00275             if(innards[i+j] != substr.innards[j]) { match = 0; break; }
00276         if(match) return 1;
00277       }
00278
```

4.26.2.19 int DynString::Length () const [inline]

Definition at line 135 of file dynstring.h.

Referenced by DSIterator::DelimMatch(), DSIterator::Position(), Replace(), AtomRegistry::SetType(), AnimFrame::Set_Output_Stream(), and DSIterator::operator int().

```
00136   { return L; }
00137
```

4.26.2.20 DynString::MAKECOMPARISONS_WITH_COMPARE (DynString)

4.26.2.21 DynString::MAKE_BIN_STREAM_OPERATORS (DynString)

4.26.2.22 DynString::MAKE_IO_STREAM_OPERATORS (DynString)

4.26.2.23 void DynString::MakeBad () [inline, private]

Definition at line 223 of file dynstring.h.

Referenced by Clear(), DynString(), Set(), Sub(), and operator=().

```
00224   { N=0; L=0; delete[] innards; innards = Bad; }
00225
```

4.26.2.24 int DynString::NonDestResize (int *nblocks*) [private]

Definition at line 81 of file dynstring.C.

Referenced by NonDestResizeToFit().

```
00082   {
00083     if(N == nblocks) return 1;
00084     if(nblocks<1) return 0;
00085     char * tmp = innards;
00086     if(!(innards = new char[nblocks*BlockSize]))
00087       { innards = tmp; return 0; }
00088
```

4.26.2.25 int DynString::NonDestResizeToFit (int *nchars*) [inline]

Definition at line 47 of file dynstring.h.

Referenced by Cat(), Chop(), Resize(), Set(), and operator[]().

```
00048   { return NonDestResize((nchars/BlockSize)+1); }
00049
```

4.26.2.26 int DynString::Output (ostream & *os*) const

Definition at line 733 of file dynstring.C.

```
00734   {
00735     if(IsBad())
00736         os << "<<BAD STRING!>>";
00737     else
00738         os << innards;
00739     return L;
00740
```

4.26.2.27 int DynString::OutputBinary (ostream & *os*) const

Definition at line 443 of file dynstring.C.

```
00444   {
00445     os <= L;
00446     if(L>0)
00447         os.write(innards, L*sizeof(char));
00448     return L;
00449
```

4.26.2.28 const DynString & DynString::Pad (char *c*, int *n*)

Definition at line 755 of file dynstring.C.

```
00756   {
00757     DynString padding = Reproduce(c,n);
00758     return Cat(padding);
00759
```

4.26.2.29 int DynString::Read (istream & *is*, bool *append* = false)

Definition at line 460 of file dynstring.C.
Referenced by Input(), and ReadLine().

```
00461   {
00462     char c;
00463     int len=0;
00464
00465     if(!append) Clear();
00466
00467     is.unsetf(ios::skipws);
00468     // Skip all the header delimiter characters.
00469     for(int done=0; !done; )
00470       {
00471         c = is.peek();
00472
00473         // See if the character matches any delimiter character.
00474         if(c == EOF || !Delimiter[(uint) c]) done = 1;
00475
00476         // Otherwise advance to the next character in the stream.
00477         else is.get();
00478       }
00479
00480     // Read in a series of consecutive non-delimiter characters.
00481     for(int done=0; !done; )
00482       {
00483         c = is.peek();
00484         if((c == EOF) || Delimiter[(uint) c]) done = 1;
```

```
00485         else
00486           {
00487             Cat(c);
00488             is.get();
00489             ++len;
00490           }
00491       }
00492
```

4.26.2.30 istream & DynString::ReadLine (istream & *is*, bool *append* = false)

Definition at line 761 of file dynstring.C.
Referenced by EatLine(), Molecule::readFasta(), Molecule::readXmol(), and Molecule::skipXmol().

```
00762   {
00763     DynString old_delim = GetDSDelim();
00764     SetDSDelim(DynString("\n"));
00765     Read(is,append);
00766     SetDSDelim(old_delim);
00767     return is;
00768
```

4.26.2.31 int DynString::Replace (const DynString & *oldstr*, const DynString & *newstr*)

Definition at line 281 of file dynstring.C.

```
00282   {
00283     int oldlen = oldstr.Length(), newlen = newstr.Length();
00284     int * match = 0;
00285     int replace=0;
00286     int total=-1;
00287
00288     if(L<1 || oldlen<1) return 0;
00289
00290     if(!(match = new int[L]))
00291         { cerr << "DynString::Replace#1\n"; raise(SIGQUIT); }
00292
00293     if(oldlen < 1) return 0;
00294
00295     // Find the number of (non-overlapping replacements we need to make):
00296     for(int i=0; i<L; ++i) match[i] = 0;
00297     for(int i=0; i<L-oldlen+1; )
00298       {
00299         if(!strncmp(innards+i, oldstr.innards, oldlen))
00300           {
00301             // Found a match!
00302             ++replace;
00303             match[i] = 1;
00304             i += oldlen;
00305           }
00306         else
00307           { match[i] = 0; ++i; }
00308       }
00309
00310     total = L + replace*(newlen-oldlen);
00311
00312     if(total<1)
00313       {
```

```
00314         delete[] match;
00315         return 0;
00316       }
00317
00318     // Copy the old string.
00319     char * oldinnards = 0;
00320     if(!(oldinnards = new char[L+1]))
00321         { cerr << "DynString::Replace#2\n"; raise(SIGQUIT); }
00322     int oldL = L;
00323     for(int i=0; i<oldL; ++i) oldinnards[i] = innards[i];
00324     oldinnards[oldL] = '\0';
00325
00326     // Make space for the new string.
00327     if(!DestResizeToFit(total))
00328         { cerr << "DynString::Replace#3\n"; raise(SIGQUIT); }
00329
00330     int j=0;
00331     for(int i=0; i<oldL; )
00332       {
00333         if(match[i])
00334           {
00335             for(int k=0; k<newlen; ++k)
00336                 innards[j++] = newstr[k];
00337             i += oldlen;
00338           }
00339         else
00340             innards[j++] = oldinnards[i++];
00341       }
00342
```

4.26.2.32 int DynString::Resize (int *newlength*) [inline]

Definition at line 49 of file dynstring.h.

```
00050   { return NonDestResizeToFit(newlength); }
00051
```

4.26.2.33 int DynString::Set (int *i*, char *c*)

Definition at line 154 of file dynstring.C.

```
00155   {
00156     if(IsBad() || i<0) return 0;
00157     if(i>=N*BlockSize-1)
00158       {
00159         if(!NonDestResizeToFit(i))
00160           {
00161             MakeBad();
00162             return 0;
00163           }
00164       }
00165     if(i>=L)
00166       {
00167         innards[L] = innards[N*BlockSize-1];
00168         L = i+1;
00169         innards[L] = '\0';
00170       }
00171
```

4.26.2.34 int DynString::Set (const char * *str*, int *num*)

Definition at line 136 of file dynstring.C.

```
00137   {
00138      if(!str || str == Bad)
00139       { MakeBad(); return 0; }
00140
00141     if(num<=0)
00142       { DestResize(1); return 0; }
00143
```

4.26.2.35 int DynString::Set (const DynString & *ds*)

Definition at line 117 of file dynstring.C.
Referenced by DynString(), and operator=().

```
00118   {
00119     if(this == &ds)
00120         return 1;
00121
00122     if(ds.L<1)
00123       { DestResize(1); return 0; }
00124
```

4.26.2.36 int DynString::Split (DynString *& *substr*) const

Definition at line 496 of file dynstring.C.

```
00497   {
00498     // Number of substrings returned;
00499     int n=0;
00500
00501     if(substr)
00502         delete[] substr;
00503     substr = 0;
00504
00505     // Count the number of substrings there are:
00506     int i=0;
00507     while(i<L)
00508       {
00509         // Skip leading stretch of delimiters:
00510         for(; i<L; ++i)
00511           {
00512             if(!Delimiter[(uint) innards[i]])
00513                 break;
00514             cout << '<' << innards[i] << '>';
00515           }
00516         // Increment the substring count if there's still
00517         // characters left in the string:
00518         ++n;
00519
00520         // Skip over this substring:
00521         for(; i<L; ++i)
00522           {
00523             if(Delimiter[(uint) innards[i]])
00524                 break;
00525             cout << '[' << innards[i] << ']';
00526           }
```

```
00527       }
00528
00529     if(n < 1)
00530         return 0;
00531
00532     // Create space for the substrings:
00533     substr = new DynString[n];
00534
00535     i=0;
00536     int j=-1;
00537     while(i<L)
00538       {
00539         // Skip all the header delimiter characters.
00540         for(i=0; i<L; ++i)
00541             // See if the character matches any delimiter.
00542             if(!Delimiter[(uint) innards[i]])
00543                 break;
00544
00545         // Read in a series of consecutive non-delimiter.
00546         substr[++j] = "";
00547         for(; i<L; ++i)
00548           {
00549             if(Delimiter[(uint) innards[i]])
00550                 break;
00551             else
00552                 substr[j].Cat(innards[i]);
00553           }
00554       }
00555
```

4.26.2.37 DynString DynString::Sub (int *i*, int *j*) const

Definition at line 236 of file dynstring.C.
Referenced by operator()().

```
00237   {
00238     DynString substr = "";
00239     if(L <= 0) return substr;
00240     if(IsBad())
00241       {
00242         substr.MakeBad();
00243         return substr;
00244       }
00245
00246     int k;
00247     if(j<i)
00248       {
00249         k = j;
00250         j = i;
00251         i = k;
00252       }
00253
00254     if(i<0) i = 0;
00255     if(j >= L) j = L-1;
00256
00257     if(!substr.DestResizeToFit(j-i+1))
00258         { cerr << "DynString::Sub\n"; raise(SIGQUIT); }
00259
```

4.26.2.38 double DynString::ToDbl () const [inline]

Definition at line 114 of file dynstring.h.
Referenced by ddGraph::Input(), and main().

```
00115   { return atof(innards); }
00116
```

4.26.2.39 int DynString::ToInt () const [inline]

Definition at line 116 of file dynstring.h.
Referenced by ddGraph::Input(), main(), Molecule::readXmol(), and Molecule::skipXmol().

```
00117   { return ((int) atoi(innards)); }
00118
```

4.26.2.40 void DynString::ToLower () [inline]

Definition at line 110 of file dynstring.h.

```
00111   { for(int i=0; i<L; ++i) innards[i] = tolower(innards[i]); }
00112
```

4.26.2.41 void DynString::ToUpper () [inline]

Definition at line 108 of file dynstring.h.
Referenced by Molecule::readFasta().

```
00109   { for(int i=0; i<L; ++i) innards[i] = toupper(innards[i]); }
00110
```

4.26.2.42 DynString DynString::operator() (int *i*, int *j*) const [inline]

Definition at line 96 of file dynstring.h.

```
00097   { return Sub(i,j); }
00098
```

4.26.2.43 const DynString & DynString::operator+= (const DynString & *ds*) [inline]

Definition at line 157 of file dynstring.h.

```
00158   { return Cat(ds); }
00159
```

4.26.2.44 const DynString & DynString::operator= (double *number*)

Definition at line 383 of file dynstring.C.

```
00384   {
00385     char * s = new char[30];
00386     sprintf(s, "%g", number);
00387     Set(s,strlen(s));
00388     delete[] s;
00389     return *this;
00390
```

4.26.2.45 const DynString & DynString::operator= (char *c*)

Definition at line 372 of file dynstring.C.

```
00373   {
00374     // Get rid of extra space if it is currently allocated.
00375     //
00376     if(!DestResizeToFit(1)) { MakeBad(); return *this; }
00377
```

4.26.2.46 const DynString & DynString::operator= (const char * *str*)

Definition at line 369 of file dynstring.C.

```
00370   { if(innards != str) Set(str,strlen(str)); return *this; }
00371
```

4.26.2.47 const DynString & DynString::operator= (const DynString & *ds*)

Definition at line 362 of file dynstring.C.

```
00363   {
00364     if(!ds.innards) return ($\ast$this);
00365     Set(ds);
00366     return *this;
00367
```

4.26.2.48 char & DynString::operator[ ] (int *i*)

Definition at line 399 of file dynstring.C.

```
00400   {
00401     if(i<0)
00402       { cerr << "DynString::operator[]\n"; raise(SIGQUIT); }
00403     if(i>=L)
00404       {
00405         NonDestResizeToFit(i+1);
00406         L = i+1;
00407         innards[L] = '\0';
00408       }
00409
```

4.26.2.49 const char & DynString::operator[] (int *i*) const

Definition at line 392 of file dynstring.C.

```
00393   {
00394     if((i>=L)||(i<0))
00395       { cerr << "DynString::operator[]\n"; raise(SIGQUIT); }
00396
```

4.26.3 Friends And Related Function Documentation

4.26.3.1 void AddDSDelim (char *delim*) [friend]

Definition at line 45 of file dynstring.C.

```
00046   { DynString::Delimiter[(uint) delim] = 1; }
00047
```

4.26.3.2 int DSprintf (DynString & *ds*, const char * *cntl*, ...) [friend]

4.26.3.3 DynString GetDSDelim () [friend]

Definition at line 51 of file dynstring.C.
Referenced by ReadLine().

```
00052   {
00053     DynString delim;
00054     delim.DestResizeToFit(256);
00055     int j=0;
00056     for(uint i=0; i<256; ++i)
00057       {
00058         if(DynString::Delimiter[i])
00059           {
00060             delim.innards[j] = (char) i;
00061             ++j;
00062           }
00063       }
00064
```

4.26.3.4 DynString IntToDS (int *i*) [friend]

Definition at line 351 of file dynstring.C.

```
00352   {
00353     char s[100]; s[0] = '\0';
00354     sprintf(s, "%i", i);
00355     DynString ds(s);
00356     return ds;
00357
```

4.26.3.5 void RemDSDelim (char *delim*) [friend]

Definition at line 47 of file dynstring.C.

```
00048   { DynString::Delimiter[(uint) delim] = 0; }
00049
```

4.26.3.6 DynString Reproduce (char *c*, int *n*) [friend]

Definition at line 745 of file dynstring.C.
Referenced by Pad().

```
00746   {
00747     DynString str;
00748     if(!str.DestResizeToFit(n)) return str;
00749     for(int i=0; i<n; ++i) str.innards[i] = c;
00750     str.innards[n] = '\0';
00751     str.L = n;
00752     return str;
00753
```

4.26.3.7 void SetDSDelim (const DynString & *delim*) [friend]

Definition at line 34 of file dynstring.C.
Referenced by ReadLine().

```
00035   {
00036     uint j;
00037     for(int i=0; i<256; ++i) DynString::Delimiter[i] = 0;
00038     for(int i=0; i<delim.L; ++i)
00039       {
00040         j = (uint) delim.innards[i];
00041         DynString::Delimiter[j] = 1;
00042       }
00043
```

4.26.3.8 DynString operator+ (DynString *x*, const DynString & *y*) [friend]

Definition at line 162 of file dynstring.h.

```
00163   { return x+=y; }
00164
```

4.26.4 Member Data Documentation

4.26.4.1 char * DynString::Bad = 0 [static]

Definition at line 16 of file dynstring.C.

4.26.4.2 const int DynString::BlockSize = 32 [static]

Definition at line 11 of file dynstring.C.

4.26.4.3 int DynString::Delimiter[256] [static]

Initial value:

```
{
```

```
  1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,1,
  1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,
  0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
}
```

Definition at line 20 of file dynstring.C.

4.26.4.4 DynString DynString::Empty = "" [static]

Definition at line 18 of file dynstring.C.

4.26.4.5 int DynString::L [private]

Definition at line 243 of file dynstring.h.

4.26.4.6 int DynString::N [private]

Definition at line 240 of file dynstring.h.

4.26.4.7 char * DynString::innards [private]

Definition at line 238 of file dynstring.h.

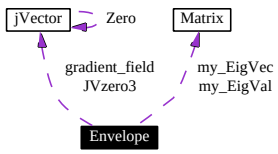
The documentation for this class was generated from the following files:

- **dynstring.h**
- **dynstring.C**

4.27 Envelope Class Reference

`#include <envelope.h>`

Collaboration diagram for Envelope:



Public Methods

- **Envelope** ()
  constructor.
- **Envelope** (const Envelope &e)
  copy constructor.
- **~Envelope** ()
  destructor.
- const double **operator()** (int x, int y, int **z**) const
  access density by cell.
- const double **operator()** (const **jVector** &J) const
  access density by position.
- const **jVector**& **gradient** (int x, int y, int **z**) const
  access gradient by cell.
- const **jVector**& **gradient** (**jVector** J) const
  access gradient.
- int **Build** (const **Molecule** &Mol, double IW_in=DEFAULT_CUBE_WIDTH)
  create envelope from current molecule.
- void **Get_Eigenvectors** (**Matrix** &EigVec, **Matrix** &EigVal) const
  writes Eigenvalues and Eigenvectors matricies by reference.
- int **WriteDensity** (ostream &outStream, const **Molecule** &Mol) const
  write out a list of atoms and their density in the env.
- int **WriteAvsTxt** (ostream &outStream) const
  write AVS density format(as text).
- int **WriteVRML** (ostream &outStream) const
  write VRML 1.0 format.

- int **WriteVRMLgr** (ostream &outStream) const
  write VRML 1.0 format with gradient arrows.
- void **DiffReport** (ostream &outStream, const Envelope &E)
  Report on differences to another env.
- int **Compare** (const Envelope &ec) const
  default compare function.
- **MAKECOMPARISONS_WITH_COMPARE** (Envelope)
- int **Read** (istream &is)
- int **ReadBinary** (istream &is)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const
- void **Calc_Eigenvectors** ()
  internal call to calc and store Eigenvectors.
- void **CalcGradient** (int d=1)
  uses the current density_array to calculate the gradient_field.
- void **Upsize** (int k)
  increases the box size by a factor k'.
- void **Blur** (int k)
  performs flat kernel convolution of 3D size (2∗kernelSize+1) per side.
- void **NormalizeMax** ()
  max max density = def's MAX_ENV_DEN' value and scale all cells accordingly.
- double **QueryDensity** (const **jVector** &pos) const
  returns density at 'pos.
- **jVector QueryGradient** (const **jVector** &pos) const
- double& **operator()** (int x, int y, int **z**)
  lvalue (for assigning density).
- **jVector**& **gradient** (int x, int y, int **z**)
  lvalue (for assigning gradient).
- int **ReadAvsTxt** (istream &in)
  read text-converted AVS density format.
- void **Env_Setup1** ()
  allocates memory.
- void **Env_Setup2** ()
  calculates meta information about data.
- void **Center** (void)
  adjust the max[xyz] and min[xyz] of the envelope to position the center of mass at the origin.

Public Attributes

- double* **density_array**

  Array of doubles storing density.

- **jVector*** **gradient_field**

  Array of jVectors storing gradient.

- int **numPoints**

  Number of cubes in Envelope.

- int **xdim**

  Dimensions in number of cubes.

- int **ydim**

  Dimensions in number of cubes.

- int **zdim**

  Dimensions in number of cubes.

- double **InputWidth**

  Real width of cubes (e.g. 3.7A).

- double **minx**

  Real position of min edges of region.

- double **miny**

  Real position of min edges of region.

- double **minz**

  Real position of min edges of region.

- double **maxx**

  Real position of max edges of region.

- double **maxy**

  Real position of max edges of region.

- double **maxz**

  Real position of max edges of region.

- double **cm_x**

  Center of mass, weighted by probabilities, (as blocks).

- double **cm_y**

  Center of mass, weighted by probabilities, (as blocks).

- double **cm_z**

  Center of mass, weighted by probabilities, (as blocks).

- double **r_cm_x**

  Center of mass, weighted by probabilities, (Real postition!).

- double **r_cm_y**

  Center of mass, weighted by probabilities, (Real postition!).

- double **r_cm_z**

  Center of mass, weighted by probabilities, (Real postition!).

- double **totalDensity**

  Total of all the density numbers in the cubes.

- int **filledPoints**

  Number of non-zero cubes in Envelope.

- **Matrix my_EigVal**

  Eigenvalues for the envelope.

- **Matrix my_EigVec**

  Eigenvectors for the envelope.

- const **jVector JVzero3**

  useful null vector.

Friends

- class **EnvelopeConstraint**

4.27.1 Constructor & Destructor Documentation

4.27.1.1 Envelope::Envelope ()

constructor.

Definition at line 3 of file envelope.C.

```
00005 {
00006   // this is an unititialized envelope.
00007
00008   xdim = ydim = zdim = -1;
00009   minx = miny = minz = maxx = maxy = maxz = 0;
00010
00011   density_array = 0;
00012   gradient_field = 0;
00013
00014   numPoints = 0;
00015   InputWidth = -1;
00016
00017
```

4.27.1.2 Envelope::Envelope (const Envelope & *env_in*)

copy constructor.

Definition at line 20 of file envelope.C.

```
00021 {
00022
00023   xdim = env_in.xdim;
00024   ydim = env_in.ydim;
00025   zdim = env_in.zdim;
00026
00027   minx = env_in.minx;
00028   miny = env_in.miny;
00029   minz = env_in.minz;
00030
00031   maxx = env_in.maxx;
00032   maxy = env_in.maxy;
00033   maxz = env_in.maxz;
00034
00035   InputWidth = env_in.InputWidth;
00036   numPoints = env_in.numPoints;
00037
00038   // allocate
00039   this->Env_Setup1();
00040
00041   // copy data
00042   for (int c = 0; c<numPoints; c++){
00043     density_array[c] = env_in.density_array[c];
00044     gradient_field[c] = env_in.gradient_field[c];
00045   }
00046
```

4.27.1.3 Envelope::~Envelope ()

destructor.

Definition at line 52 of file envelope.C.

```
00052                     {
00053
00054   if (density_array)
00055     delete [] density_array;
00056
00057   if (gradient_field){
00058     delete [] gradient_field;
00059   }
00060
```

4.27.2 Member Function Documentation

4.27.2.1 void Envelope::Blur (int *k*)

performs flat kernel convolution of 3D size (2*kernelSize+1) per side.

Definition at line 714 of file envelope.C.

Referenced by EnvelopeConstraint::PreSetup().

```
00714                      {
00715
```

```
00716   //  cerr << "------------ Blur Envelope Start    --------" << endl;
00717
00718   if (k < 1){
00719     DIE("kernel size must be > 0");
00720   }
00721
00722   int k2 = 2 * k;
00723   int kernelsize = (k2+1)*(k2+1)*(k2+1); // number of cells in the kernel
00724
00725   int new_numPoints = (xdim+k2)*(ydim+k2)*(zdim+k2);
00726   int new_xdim = xdim + k2;
00727   int new_ydim = ydim + k2;
00728   int new_zdim = zdim + k2;
00729
00730   double o_kernelsize = 1./(double)kernelsize;
00731   double d = InputWidth * k;
00732   double * new_density_array = new double[new_numPoints]; // not sparse -- saving lots of zeros here. mem wasted.  FI
00733   for (int l=0;l<new_numPoints;l++){new_density_array[l] =0;}
00734   jVector * new_gradient_field = new jVector[new_numPoints];
00735             // need to fix gradients after resize on blur. -- consider RESIZE function.
00736
00737   double spread_val;  // density value to be "spread" to its neighbors
00738
00739   // cycle over old array, processing cells with data
00740   int zh,yh,xh;
00741   for (zh =0;zh<zdim ;zh++){
00742     for (yh =0;yh<ydim ;yh++){
00743       for (xh =0;xh<xdim ;xh++){
00744         spread_val = ($\ast$this)(xh,yh,zh);
00745
00746         if (spread_val > 0) {
00747
00748           // spread out value in cell across kernel (normalized to o_kernelsize)
00749           if (spread_val == 1) {
00750             spread_val = o_kernelsize;
00751           } else {
00752             spread_val = spread_val * o_kernelsize;
00753           }
00754
00755           for (int kz = 0; kz<(k2+1);kz++){
00756             for (int ky = 0; ky<(k2+1);ky++){
00757               for (int kx = 0; kx<(k2+1);kx++){
00758                 new_density_array[\
00759                                   (xh+kx)  \
00760                                   +  ((yh*new_xdim)+(ky*new_xdim))  \
00761                                   +  ((zh*new_xdim*new_ydim)+(kz*new_ydim*new_xdim)) \
00762                 ] += spread_val;
00763
00764               }  //kernel x
00765             }    //kernel y
00766           }      //kernel z
00767         } // skip 0 cells in old matrix.
00768       }   //x
00769     }     //y
00770   }       //z
00771
00772   //  resize to fit k more cells per side (d is the real size)
00773
00774   minx -= d;
00775   miny -= d;
00776   minz -= d;
00777
00778   maxx += d;
00779   maxy += d;
00780   maxz += d;
```

```
00781
00782
00783   xdim = new_xdim;
00784   ydim = new_ydim;
00785   zdim = new_zdim;
00786
00787   if (density_array && gradient_field) {
00788     delete [] density_array;
00789     density_array = new_density_array;
00790
00791     delete [] gradient_field;
00792     gradient_field = new_gradient_field;  // empty
00793
00794     numPoints = new_numPoints;
00795
00796   } else {
00797     DIE("Called Blur on Envelope without intializing internal datastructures.");
00798   }
00799
```

4.27.2.2 int Envelope::Build (const Molecule & *Mol*, double *IW_in* = DEFAULT_CUBE_WIDTH)

create envelope from current molecule.

Definition at line 248 of file envelope.C.

Referenced by jdOcc_Count::run().

```
00249 {
00250
00251    //  here we need to build an envelope to describe the CM current molecule
00252
00253   InputWidth = IW_in;
00254   Mol.Bounds(minx,miny,minz,maxx,maxy,maxz);
00255   // figure out dimensions
00256
00257   xdim = 1 + int ((maxx-minx)/InputWidth);
00258   ydim = 1 + int ((maxy-miny)/InputWidth);
00259   zdim = 1 + int ((maxz-minz)/InputWidth);
00260
00261   maxx = minx + InputWidth*xdim; // reset to match
00262   maxy = miny + InputWidth*ydim;
00263   maxz = minz + InputWidth*zdim;
00264
00265   numPoints = xdim*ydim*zdim;
00266
00267   this->Env_Setup1();
00268
00269   int zh, yh, xh;
00270   for (zh =0;zh<zdim ;zh++){
00271     for (yh =0;yh<ydim ;yh++){
00272       for (xh =0;xh<xdim ;xh++){
00273         ($\ast$this)(xh,yh,zh) = 0;
00274       }
00275     }
00276   }
00277
00278   int n = Mol.Number();
00279   jVector center;
00280   double radius;
00281
00282   RanDbl RD;
```

```
00283   //  double cenx, ceny, cenz;
00284   double cenx, ceny, cenz;
00285   register double x, y, z;
00286   int x_cell,y_cell,z_cell;
00287   int N = 100000;
00288
00289   // what we really want to count is how much "atom" there
00290   // is overlapping each box.
00291
00292   // we exect 1/frac points per atom to be counted
00293   double frac = 6.02/(N * PI);
00294
00295   // except in cases where the box size is smaller then the atom size
00296   // in this case we want these boxes to have density of 1 so we have
00297   // to adjust the 'frac' accordingly
00298
00299   // assuming carbon VDW radius of 1.4 A here:
00300   if (InputWidth < 2.8) {
00301     cerr << "Envelope::Build->adjusting fraction for small gridsize" << endl;
00302     double temp1 = (1.4/InputWidth);
00303     frac = 8 * temp1* temp1* temp1 / N;
00304
00305   }
00306   WARN_IF(frac > 0.01, "May need to increase counts for good statistics");
00307     //  cerr << "frac = " << frac << endl;
00308
00309   for (int i =0; i < n; i++){
00310     cerr << ".";
00311     center = Mol(i);
00312     radius = Mol.getAtom(i).VDW();
00313
00314     cenx = center[0];
00315     ceny = center[1];
00316     cenz = center[2];
00317
00318     for (int j = 0;j<N;j++){
00319       x = RD.Get(-1,1);       y = RD.Get(-1,1);       z = RD.Get(-1,1);
00320       // XYZ ON (0,1) range
00321       if (x*x+y*y+z*z < 1){
00322         // point is inside unit sphere
00323         x = cenx + (x * radius);
00324         y = ceny + (y * radius);
00325         z = cenz + (z * radius);
00326
00327         x_cell = (int) floor((x-minx)/InputWidth);
00328         y_cell = (int) floor((y-miny)/InputWidth);
00329         z_cell = (int) floor((z-minz)/InputWidth);
00330
00331         ($\ast$this)(x_cell,y_cell,z_cell) = ($\ast$this)(x_cell,y_cell,z_cell) + frac;
00332
00333       }// inside sphere?
00334     } // gen points
00335   }
00336
00337   cerr << endl;
00338
00339   for (zh =0;zh<zdim ;zh++){
00340     for (yh =0;yh<ydim ;yh++){
00341       for (xh =0;xh<xdim ;xh++){
00342         if (($\ast$this)(xh,yh,zh) > MAX_ENV_DEN)
00343           ($\ast$this)(xh,yh,zh) = MAX_ENV_DEN;
00344       }
00345     }
00346   }
00347
```

4.27.2.3 void Envelope::CalcGradient (int $d = 1$)

uses the current density_array to calculate the gradient_field.

Definition at line 440 of file envelope.C.
Referenced by Blur(), ReadAvsTxt(), and Upsize().

```
00440                                {
00441
00442   if (d < 1){
00443     DIE ("gradient distance must be > 0");
00444   }
00445
00446   double den;
00447   double val;
00448   jVector direction;
00449   int x,y,z;
00450   jVector J;
00451   int zh,yh,xh;
00452
00453   for (zh =0;zh<zdim ;zh++){
00454     for (yh =0;yh<ydim ;yh++){
00455       for (xh =0;xh<xdim ;xh++){
00456
00457         den =($\ast$this)(xh,yh,zh);
00458
00459         // setting gradient at xh,yh,zh
00460         (($\ast$this).gradient(xh,yh,zh)).zero();
00461
00462         for (int kz = (-1*d); kz<=d;kz++){
00463           for (int ky = (-1*d); ky<=d;ky++){
00464             for (int kx = (-1*d); kx<=d;kx++){
00465
00466
00467
00468               // add-from coords
00469               x = xh+kx;
00470               y = yh+ky;
00471               z = zh+kz;
00472
00473               if ((x >= 0) && (x < xdim) &&
00474                   (y >= 0) && (y < ydim) &&
00475                   (z >= 0) && (z < zdim))  {
00476                 // bounds OK
00477
00478                 J.zero();
00479                 if ((val = ($\ast$this)(x,y,z))){ //sic
00480                   J[0] = kx*val;
00481                   J[1] = ky*val;
00482                   J[2] = kz*val;
00483
00484                   J.norm_unit(); // J div by magnitude squared
00485
00486                   (($\ast$this).gradient(xh,yh,zh)) = (($\ast$this).gradient(xh,yh,zh)) + J;
00487                 } // else add nothing -- no density here
00488
00489               } // bounds ok on add-from box
00490
00491               }  //kernel x
00492             }    //kernel y
00493           }      //kernel z
00494
00495       }   //x
00496     }     //y
```

```
00497   }
00498
```

4.27.2.4 void Envelope::Calc_Eigenvectors (void)

internal call to calc and store Eigenvectors.

Definition at line 510 of file envelope.C.

Referenced by Env_Setup2().

```
00510                                {
00511
00512   Matrix Spread(3,3); // real symmetric
00513   Spread.zero();
00514
00515   //  jVector tempV;
00516   double val;
00517   // Calc spread
00518   double dx, dy, dz = 0;
00519
00520
00521   for (int zh =0;zh<zdim ;zh++){
00522     for (int yh =0;yh<ydim ;yh++){
00523       for (int xh =0;xh<xdim ;xh++){
00524
00525         val = ($\ast$this)(xh,yh,zh);
00526
00527         if (val > 0) {
00528
00529 //        dx = (xh - cm_x)*InputWidth*val/totalDensity;
00530 //        dy = (yh - cm_y)*InputWidth*val/totalDensity;
00531 //        dz = (zh - cm_z)*InputWidth*val/totalDensity;
00532           dx = (xh - cm_x)*InputWidth*val;
00533           dy = (yh - cm_y)*InputWidth*val;
00534           dz = (zh - cm_z)*InputWidth*val;
00535
00536
00537           Spread (1,1) += dx*dx/filledPoints;
00538           Spread (2,1) += dx*dy/filledPoints;
00539           Spread (2,2) += dy*dy/filledPoints;
00540           Spread (3,1) += dx*dz/filledPoints;
00541           Spread (3,2) += dy*dz/filledPoints;
00542           Spread (3,3) += dz*dz/filledPoints;
00543 //        Spread (1,1) += dx*dx;
00544 //        Spread (2,1) += dx*dy;
00545 //        Spread (2,2) += dy*dy;
00546 //        Spread (3,1) += dx*dz;
00547 //        Spread (3,2) += dy*dz;
00548 //        Spread (3,3) += dz*dz;
00549
00550         } // end if
00551       } // x
00552     } // y
00553   }
00554
```

4.27.2.5 void Envelope::Center (void)

adjust the max[xyz] and min[xyz] of the envelope to position the center of mass at the origin.

Definition at line 111 of file envelope.C.
Referenced by EnvelopeConstraint::Read().

```
00111                           {
00112
00113 #ifdef NOISY_CONSTRAINTS
00114   WARN("Centering Envelope data on (0,0,0)");
00115   cerr << "original center_of_mass: (" << r_cm_x << "," << r_cm_y << "," << r_cm_z << ")" << endl;
00116 #endif
00117   maxx -= r_cm_x;
00118   minx -= r_cm_x;
00119   maxy -= r_cm_y;
00120   miny -= r_cm_y;
00121   maxz -= r_cm_z;
00122   minz -= r_cm_z;
00123
00124   // re calc metadata
00125   Env_Setup2();
00126
00127 #ifdef NOISY_CONSTRAINTS
00128   cerr << "new      center_of_mass: (" << r_cm_x << "," << r_cm_y << "," << r_cm_z << ")" << endl;
00129   cerr << "done centering envelope." << endl;
00130 #endif
00131
```

4.27.2.6 int Envelope::Compare (const Envelope & *e*) const [inline]

default compare function.
Definition at line 190 of file envelope.h.

```
00191 {
00192   DIE("Envelope Compare not Implemented");
00193   return 0;
00194
```

4.27.2.7 void Envelope::DiffReport (ostream & *outStream*, const Envelope & *E*)

Report on differences to another env.
Definition at line 352 of file envelope.C.
Referenced by main().

```
00352                                                                   {
00353
00354   if (xdim != E.xdim || ydim != E.ydim || zdim != E.zdim ){
00355     outStream << "dimensions differ:\n" << xdim << " " <<ydim << " " <<zdim  << endl
00356               << E.xdim << " " <<E.ydim << " " <<E.zdim  << endl;
00357     outStream << "aborting";
00358     return;
00359   } else {
00360     outStream << "dimensions OK " << endl;
00361   }
00362
00363   double THRESHOLD = 0.01;
00364
00365   if (fabs(r_cm_x-E.r_cm_x)>THRESHOLD ||
00366       fabs(r_cm_y-E.r_cm_y)>THRESHOLD ||
00367       fabs(r_cm_z-E.r_cm_z)>THRESHOLD ){
```

```
00368     outStream << "COM differs: " << endl
00369               << r_cm_x << " " << r_cm_y << " " << r_cm_z << endl
00370               << E.r_cm_x << " " << E.r_cm_y << " " << E.r_cm_z << endl;
00371   } else {
00372     outStream << "COM OK " << endl;
00373   }
00374
00375   if (fabs(totalDensity -E.totalDensity) > 50* THRESHOLD)
00376     outStream << "totalDensity differs: " << totalDensity << "        " << E.totalDensity << endl;
00377   else {
00378     outStream << "totalDensity OK " << endl;
00379   }
00380
00381   if (abs(filledPoints - E.filledPoints) > 2)
00382     outStream << "filledPoints differs: " << filledPoints << "        " << E.filledPoints << endl;
00383   else {
00384     outStream << "filledPoints OK " << endl;
00385   }
00386
00387   double totalError = 0;
00388   double maxError = 0;
00389   int diff_count = 0;
00390   double diff;
00391
00392
00393   int zh,yh,xh;
00394
00395   for (zh =0;zh<zdim ;zh++){
00396     for (yh =0;yh<ydim ;yh++){
00397       for (xh =0;xh<xdim ;xh++){
00398         diff = fabs ((*this)(xh,yh,zh) - E(xh,yh,zh)) ;
00399         totalError += diff;
00400         if (diff > maxError)
00401           maxError = diff;
00402         if (diff > THRESHOLD) {
00403           diff_count++;
00404         }
00405       }
00406     }
00407   }
00408
```

4.27.2.8 void Envelope::Env_Setup1 ()

allocates memory.

Definition at line 63 of file envelope.C.

Referenced by Build(), Envelope(), and ReadAvsTxt().

```
00063                           {
00064
00065   density_array = new double[numPoints]; // not sparse -- saving lots of zeros here. mem wasted.  FIX
00066   //  cerr << " Allocating Envelope for "<< numPoints << endl;
00067   gradient_field = new jVector[numPoints];
00068
```

4.27.2.9 void Envelope::Env_Setup2 ()

calculates meta information about data.

Definition at line 70 of file envelope.C.
Referenced by Blur(), Center(), Envelope(), ReadAvsTxt(), and Upsize().

```
00070                           {
00071
00072   double val;
00073   int zh, yh, xh; // blocks
00074
00076   filledPoints =  0;
00077   totalDensity = cm_z = cm_y = cm_x = 0;
00078
00079   // first calc in blocks, then translate to real pos.
00080   for ( zh =0;zh<zdim ;zh++){
00081     for ( yh =0;yh<ydim ;yh++){
00082       for ( xh =0;xh<xdim ;xh++){
00083         val = ($\ast$this)(xh,yh,zh) ;
00084
00085         if (val > 0) {
00086           filledPoints++;
00087           totalDensity += val;
00088
00089
00090           cm_x += xh * val;
00091           cm_y += yh * val;
00092           cm_z += zh * val;
00093
00094         }
00095       }
00096     }
00097   }
00098
```

4.27.2.10 void Envelope::Get_Eigenvectors (Matrix & *EigVal*, Matrix & *EigVec*) const

writes Eigenvalues and Eigenvectors matricies by reference.

Definition at line 502 of file envelope.C.
Referenced by EnvelopeConstraint::FindEnvelopeEigenvectors().

```
00502                                                                   {
00503
00504   EigVec = my_EigVec;
00505   EigVal = my_EigVal;
00506
00507
```

4.27.2.11 Envelope::MAKECOMPARISONS_WITH_COMPARE (Envelope)

4.27.2.12 void Envelope::NormalizeMax ()

max max density = def's MAX_ENV_DEN' value and scale all cells accordingly.

Definition at line 813 of file envelope.C.
Referenced by Blur().

```
00813                       {
```

```
00814
00815   double spread_val;
00816   double max_grid_val = -INF;
00817
00818   // find max
00819   for (int zh =0;zh<zdim ;zh++){
00820     for (int yh =0;yh<ydim ;yh++){
00821       for (int xh =0;xh<xdim ;xh++){
00822         spread_val = ($\ast$this)(xh,yh,zh);
00823
00824         if (spread_val > max_grid_val) {
00825           max_grid_val = spread_val;
00826         }
00827       }
00828     }
00829   }
00830
00831   double o_max_grid_val = MAX_ENV_DEN / max_grid_val;
00832
00833   // reset
00834   for (int zh =0;zh<zdim ;zh++){
00835     for (int yh =0;yh<ydim ;yh++){
00836       for (int xh =0;xh<xdim ;xh++){
00837         spread_val = ($\ast$this)(xh,yh,zh);
00838
00839         ($\ast$this)(xh,yh,zh) = spread_val * o_max_grid_val;
00840         //      cerr << spread_val << "   " << spread_val * o_max_grid_val << endl;
00841       }
00842     }
00843   }
00844
```

4.27.2.13 int Envelope::Output (ostream & *os*) const

Definition at line 215 of file envelope.C.

```
00216 {
00217
00218   os << "\nxdim = " << xdim
00219      << "\nydim = " << ydim
00220      << "\nzdim = " << zdim
00221
00222      << "\nminx = " << minx
00223      << "\nminy = " << miny
00224      << "\nminz = " << minz
00225
00226      << "\nmaxx = " << maxx
00227      << "\nmaxy = " << maxy
00228      << "\nmaxz = " << maxz << endl;
00229
00230   os <<   "\nnumPoints = "  << numPoints
00231      <<   "\nInputWidth = " << InputWidth << endl<< endl;
00232
00233   int zh, yh, xh;
00234   for (zh =0;zh<zdim ;zh++){
00235     for (yh =0;yh<ydim ;yh++){
00236       for (xh =0;xh<xdim ;xh++){
00237         os << ($\ast$this)(xh,yh,zh) << " ";
00238       }
00239         os << endl;
00240     }
00241     os << "--" << endl;
00242   }
```

```
00243
```

4.27.2.14 int Envelope::OutputBinary (ostream & *os*) const

Definition at line 424 of file envelope.C.

```
00425 {
00426   // NOT IMPLEMENTED
00427   //  os <= supportA <= measured <= variance;
00428   return 2;
00429
```

4.27.2.15 double Envelope::QueryDensity (const jVector & *pos*) const

returns density at 'pos.

Definition at line 849 of file envelope.C.

```
00849                                                {
00850
00851   int xh = (int)((pos[0] - minx)/InputWidth);
00852   int yh = (int)((pos[1] - miny)/InputWidth);
00853   int zh = (int)((pos[2] - minz)/InputWidth);
00854
00855   return ($\ast$this)(xh,yh,zh);
00856   return 0;
00857
```

4.27.2.16 jVector Envelope::QueryGradient (const jVector & *pos*) const

4.27.2.17 int Envelope::Read (istream & *is*)

Definition at line 418 of file envelope.C.

Referenced by EnvelopeConstraint::Read(), and main().

```
00419 {
00420   // assume avs file now, later check here and call appropriate read-in function.
00421   return ReadAvsTxt(is);
00422
```

4.27.2.18 int Envelope::ReadAvsTxt (istream & *inStream*)

read text-converted AVS density format.

Definition at line 1032 of file envelope.C.

Referenced by Read().

```
01032                                          {
01033
01034
01035   //  cerr << "------------ READ Envelope start    --------" << endl;
01036
01037   if (density_array) {
```

```
01038     DIE("Envelope Read Error! It appears you have already read in density data into this envelope!");
01039   }
01040
01041   char d[255];  // used for reading header lines
01042
01043   inStream.getline(d,255); // # AVS
01044   inStream.getline(d,255); // ndim
01045
01046   //dim1
01047   inStream.ignore(5);
01048   inStream >>  xdim;
01049   //dim2
01050   inStream.ignore(6); // +\n
01051   inStream >>  ydim;
01052   //dim3
01053   inStream.ignore(6); // +\n
01054   inStream >>  zdim;
01055
01056   //  cerr << "dim: " << xdim << ", "<< ydim << ", "<< zdim << endl;
01057
01058   inStream.getline(d,255); // \n
01059
01060   inStream.getline(d,255); // nspace
01061   inStream.getline(d,255); // vclean
01062   inStream.getline(d,255); // data
01063   inStream.getline(d,255); // field
01064
01065   // mins
01066   inStream.ignore (9);
01067   inStream >>  minx >> miny >> minz;
01068
01069   //  cerr << "min: " <<  minx << ", " << miny << ", " << minz << endl;
01070
01071   // maxs
01072   inStream.ignore (10); // +\n
01073   inStream >>  maxx >> maxy >> maxz;
01074
01075   //  cerr << "max: "  <<  maxx << ", " << maxy << ", " << maxz << endl;
01076
01077   inStream.getline(d,255); // nspace
01078   inStream.getline(d,255); // # end of header
01079
01080   InputWidth = (maxx-minx)/xdim;
01081
01082   if (InputWidth > 50 || InputWidth < 0.01){
01083     cerr << "Input Grid Width is "<< InputWidth << " :Invalid.  Quitting." << endl;
01084     DIE("");
01085   }
01086
01087   if ((fabs(InputWidth - (maxy-miny)/ydim) > 0.01) ||
01088       (fabs(InputWidth - (maxz-minz)/zdim) > 0.01)) {
01089     cerr << "AVS Input is not cubic" << endl;
01090     cerr << "dim: " << xdim << ", "<< ydim << ", "<< zdim << endl;
01091     cerr << "min: " <<  minx << ", " << miny << ", " << minz << endl;
01092     cerr << "max: "  <<  maxx << ", " << maxy << ", " << maxz << endl;
01093     DIE("");
01094   }
01095
01096   double val;
01097   int readCount = 0;
01098   //  jVector Pos(3,0.0);
01099
01100   numPoints = xdim*ydim*zdim;
01101
01102   if (numPoints >= INT_MAX || numPoints < 0){
```

```
01103     cerr << "Envelope size error!\n\n Envelope Class uses signed ints to index data, so "<< INT_MAX << " is the maximu
01104     DIE("");
01105   }
01106
01107   this->Env_Setup1();
01108
01109   int zh, yh, xh;
01110   for ( zh =0;zh<zdim ;zh++){
01111     for ( yh =0;yh<ydim ;yh++){
01112       for ( xh =0;xh<xdim ;xh++){
01113         inStream >> val;
01114
01115         if (val > 0) {
01116           totalDensity += val;
01117         }
01118
01119         if (!inStream.good()){
01120           cerr << "Envelope reading error!\n\n read "<< readCount << " entries, expected "<< numPoints << endl;
01121           DIE("");
01122         }
01123
01124         ($\ast$this)(xh,yh,zh) = val;             // not sparse -- saving lots of zeros here. mem wasted.  FIX
01125         readCount++;
01126       }
01127     }
01128   }
01129
```

4.27.2.19 int Envelope::ReadBinary (istream & *is*)

Definition at line 431 of file envelope.C.

```
00432 {
00433   // NOT IMPLEMENTED
00434   //  is >= supportA[0] >= supportA[1] >= measured >= variance;
00435   return 2;
00436
```

4.27.2.20 void Envelope::Upsize (int *k*)

increases the box size by a factor k'.

Definition at line 569 of file envelope.C.

Referenced by EnvelopeConstraint::PreSetup().

```
00569                       {
00570
00571 //   if (k != 2 && k != 3 && k!=4){
00572 //     DIE("upsize k factor must be 2,3 or 4");
00573 //   }
00574    if (k != 2){
00575      DIE("upsize k factor must be 2");
00576    }  // 3 and 4 require more login in resetting real bounds.
00577
00578   int new_xdim,new_ydim,new_zdim;
00579   // figure out new bounds
00580
00581   int err;
00582   err = (xdim%k ? 1: 0);
00583   new_xdim = xdim/k+err;
```

```
00584
00585   err = (ydim%k ? 1: 0);
00586   new_ydim = ydim/k+err;
00587
00588   err = (zdim%k ? 1: 0);
00589   new_zdim = zdim/k+err;
00590
00591
00592   int new_numPoints = new_xdim * new_ydim * new_zdim;
00593   double new_InputWidth = k * InputWidth;
00594
00595   // begin: this whole section is more complex for k = 3 or 4
00596   //  double new_minx, new_miny, new_minz;
00597   double new_maxx, new_maxy, new_maxz;
00598
00599   new_maxx = minx + new_InputWidth*new_xdim;
00600   new_maxy = miny + new_InputWidth*new_ydim;
00601   new_maxz = minz + new_InputWidth*new_zdim;
00602   // end:   this whole section is more complex for k = 3 or 4
00603
00604   double * new_density_array = new double[new_numPoints]; // not sparse
00605   for (int l=0;l<new_numPoints;l++){new_density_array[l] =0;}
00606   jVector * new_gradient_field = new jVector[new_numPoints];
00607   for (int l=0;l<new_numPoints;l++){new_gradient_field[l].zero();}
00608
00609   double old_val;  // old density value
00610
00611   // cycle over old array, processing cells with data
00612   int zh,yh,xh;
00613   int new_index;
00614   int new_index_x, new_index_y, new_index_z;
00615   for (zh =0;zh<zdim ;zh++){
00616     //    new_index_z = (zh/k+(zh%k?1:0))*new_ydim*new_zdim;
00617     new_index_z = (zh/k)*new_ydim*new_xdim;
00618     for (yh =0;yh<ydim ;yh++){
00619       //      new_index_y = (yh/k+(yh%k?1:0))*new_xdim;
00620       new_index_y = (yh/k)*new_xdim;
00621       for (xh =0;xh<xdim ;xh++){
00622         //      new_index_x = xh/k+(xh%k?1:0);
00623         new_index_x = xh/k;
00624         old_val = ($\ast$this)(xh,yh,zh);
00625
00626 //      cerr << " xh=" << xh
00627 //           << " yh=" << yh
00628 //           << " zh=" << zh << endl;
00629 //
00630 //      cerr << " new_index_x=" << new_index_x
00631 //           << " new_index_y=" << new_index_y
00632 //           << " new_index_z=" << new_index_z << endl;
00633 //
00634 //      int aa = (xh)+(yh)*xdim+(zh)*xdim*ydim ;
00635 //      cerr << " old index = " << aa << " of " << numPoints << endl;
00636 //
00637         new_index = new_index_x+new_index_y+new_index_z;
00638 //      cerr << " new index = " << new_index << " of " << new_numPoints << endl << endl;
00639 //
00640
00641         if (old_val > 0) {
00642           old_val = old_val / (k*k*k);
00643
00644           if (new_index > new_numPoints) {
00645             DIE ("ACK");
00646           }
00647           new_density_array[new_index] += old_val;
00648         }
```

```
00649       }
00650     }
00651   }
00652
00653   maxx = new_maxx;
00654   maxy = new_maxy;
00655   maxz = new_maxz;
00656
00657 //   minx = new_minx;
00658 //   miny = new_miny;
00659 //   minz = new_minz;
00660
00661   xdim = new_xdim;
00662   ydim = new_ydim;
00663   zdim = new_zdim;
00664
00665   if (density_array && gradient_field) {
00666     delete [] density_array;
00667     density_array = new_density_array;
00668     new_density_array = 0;
00669
00670     delete [] gradient_field;
00671     gradient_field = new_gradient_field;  // empty
00672     new_gradient_field = 0;
00673
00674     InputWidth = new_InputWidth;
00675     numPoints = new_numPoints;
00676
00677   } else {
00678     DIE("Called Upsize on Envelope without intializing internal datastructures.");
00679   }
00680
```

4.27.2.21 int Envelope::WriteAvsTxt (ostream & *outStream*) const

write AVS density format(as text).

Definition at line 1003 of file envelope.C.

Referenced by jdOcc_Count::run().

```
01003                                                  {
01004
01005   outStream << "envelope: # AVS field file" << endl;
01006   outStream << "ndim=3" << endl;
01007   outStream << "xdim= " << xdim << endl;
01008   outStream << "ydim= " << ydim << endl;
01009   outStream << "zdim= " << zdim << endl;
01010   outStream << "nspace=3" << endl;
01011   outStream << "veclen=1" << endl;
01012   outStream << "data=float" << endl;
01013   outStream << "field=uniform" << endl;
01014   outStream << "min_ext= " << minx << " "  << miny  << " " << minz << endl;
01015   outStream << "max_ext= " << maxx  << " " << maxy  << " " << maxz << endl;
01016   outStream << "# end of header" << endl;
01017
01018   int zh, yh, xh;
01019   for (zh =0;zh<zdim ;zh++){
01020     for (yh =0;yh<ydim ;yh++){
01021       for (xh =0;xh<xdim ;xh++){
01022         outStream << ($\ast$this)(xh,yh,zh) << " ";
01023
01024       }
```

```
01025     }
01026   }
01027
```

4.27.2.22 int Envelope::WriteDensity (ostream & *outStream*, const Molecule & *Mol*) const

write out a list of atoms and their density in the env.

Definition at line 988 of file envelope.C.

Referenced by jdOcc_Count::run().

```
00988                                                                    {
00989
00990   jVector center;
00991
00992   int n = Mol.Number();
00993   for (int i=0;i<n;i++){
00994
00995     center = Mol(i);
00996     outStream << ($\ast$this)(center)  << ": " << i << endl;
00997
00998   }
00999
```

4.27.2.23 int Envelope::WriteVRML (ostream & *outStream*) const

write VRML 1.0 format.

Definition at line 859 of file envelope.C.

Referenced by EnvelopeConstraint::WriteVRML_OrigData(), and jdOcc_Count::run().

```
00859                                                {
00860
00861   outStream << "#VRML V1.0 ascii" << endl;
00862
00863   double x,y,z;
00864   double i_half = InputWidth/2;
00865   double val, t;
00866
00867   int zh, yh, xh;
00868   for (zh =0;zh<zdim ;zh++){
00869     for (yh =0;yh<ydim ;yh++){
00870       for (xh =0;xh<xdim ;xh++){
00871         val = ($\ast$this)(xh,yh,zh);
00872
00873
00874         if (val> 0) {
00875           t  = 0.5-val/2;
00876
00877           x = minx + i_half + xh * InputWidth;
00878           y = miny + i_half + yh * InputWidth;
00879           z = minz + i_half + zh * InputWidth;
00880
00881
00882
00883           outStream << "Separator{" << endl;
00884           if (val > .9)
00885             outStream << "Material{specularColor 0.8 0.8 0.8 diffuseColor 1 0.2 0.2 transparency 0.1" \
```

```
00886                     <<"}" << endl;
00887           else if (val > .5)
00888             outStream << "Material{specularColor 0.5 0.5 0.5 diffuseColor 0.2 0.8 0.2 transparency " \
00889                       << t <<"}" << endl;
00890           else
00891             outStream << "Material{specularColor 0.2 0.2 0.2 diffuseColor 0.3 0.3 0.3 transparency " \
00892                       << t <<"}" << endl;
00893
00894           outStream << "Transform{translation "<< x << " " << y << " "\
00895                     << z << " " <<"}" << endl;
00896           outStream << "Cube{width " << InputWidth << " height " \
00897                     << InputWidth << " depth " << InputWidth <<"}" << endl;
00898           outStream << "}" << endl;
00899
00900         }
00901
00902       }
00903     }
00904   }
00905
```

4.27.2.24 int Envelope::WriteVRMLgr (ostream & *outStream*) const

write VRML 1.0 format with gradient arrows.

Definition at line 910 of file envelope.C.

```
00910                                                    {
00911
00912   outStream << "#VRML V1.0 ascii\nSeparator{" << endl;
00913
00914   double x,y,z;
00915   double xr,yr,zr, rad;
00916   double i_half = InputWidth/2;
00917   double val, t;
00918   jVector G;
00919   int scale;
00920
00921
00922   int zh, yh, xh;
00923   for (zh =0;zh<zdim ;zh++){
00924     for (yh =0;yh<ydim ;yh++){
00925       for (xh =0;xh<xdim ;xh++){
00926         val = ($\ast$this)(xh,yh,zh);
00927         G = gradient(xh,yh,zh);
00928
00929         if (val> 0) {
00930           t  = 0.5-val/2;
00931
00932           x = minx + i_half + xh * InputWidth;
00933           y = miny + i_half + yh * InputWidth;
00934           z = minz + i_half + zh * InputWidth;
00935
00936           xr = G[2];
00937           yr = 0;
00938           zr = -1 * G[0];
00939           if((rad = G.mag())) // sic
00940             rad =  acos (G[1]/(rad));
00941
00942           scale = ((int) i_half)+1;
00943
00944           outStream << " Separator{" << endl;
00945
```

```
00946           outStream << "  Transform{translation "<< x << " " << y << " "\
00947                     << z << " " <<"}" << endl;
00948
00949           outStream << "  Separator{" << endl;
00950           if (val > .9)
00951             outStream << "   Material{specularColor 0.8 0.8 0.8 diffuseColor 1 0.2 0.2 transparency 0.1" \
00952                       <<"}" << endl;
00953           else if (val > .5)
00954             outStream << "   Material{specularColor 0.5 0.5 0.5 diffuseColor 0.2 0.8 0.2 transparency " \
00955                       << t <<"}" << endl;
00956           else
00957             outStream << "   Material{specularColor 0.2 0.2 0.2 diffuseColor 0.3 0.3 0.3 transparency " \
00958                       << t <<"}" << endl;
00959
00960           outStream << "   Cube{width " << InputWidth << " height " \
00961                     << InputWidth << " depth " << InputWidth <<"}" << endl;
00962           outStream << "  }" << endl;
00963
00964           outStream << "  Separator{" << endl;
00965
00966           outStream << "   Material {diffuseColor 0 0 1} " << endl;
00967           outStream << "   Transform {rotation " << xr << " "<< yr << " "<< zr <<" "<< rad <<"}" << endl;
00968           outStream << "   # gradient vector =" << G   << endl;
00969           outStream << "   Scale {scaleFactor " << scale <<" " << scale <<" " << scale <<"}" << endl;
00970           outStream << "   Cylinder {radius  .2 height  1 }" << endl;
00971           outStream << "   Transform {translation 0 1 0}" << endl;
00972           outStream << "   Cone { parts SIDES bottomRadius .2 height 1}" << endl;
00973
00974           outStream << "  }" << endl;
00975
00976           outStream << " }" << endl;
00977
00978         }
00979
00980       }
00981     }
00982   }
00983
00984   outStream << "}
00985
```

4.27.2.25 jVector & Envelope::gradient (int x, int y, int z)

lvalue (for assigning gradient).

Definition at line 135 of file envelope.C.

```
00135                                       {
00136   if ((x < 0) || (x >= xdim) ||
00137       (y < 0) || (y >= ydim) ||
00138       (z < 0) || (z >= zdim))  {
00139     cerr << "Envelope gradient access out of bounds.  I got "
00140          << x << ", "<< y << ", "<< z << endl;
00141     raise (SIGQUIT);
00142
00143   }
00144
```

4.27.2.26 const jVector & Envelope::gradient (jVector J) const

access gradient.

Definition at line 164 of file envelope.C.

```
00164                                    {
00165   int x,y,z;
00166
00167   // check this
00168   x = (int) floor((J[0]-minx)/InputWidth);
00169   y = (int) floor((J[1]-miny)/InputWidth);
00170   z = (int) floor((J[2]-minz)/InputWidth);
00171
00172   return gradient(x,y,z);
00173
```

4.27.2.27 const jVector & Envelope::gradient (int *x*, int *y*, int *z*) const

access gradient by cell.

Definition at line 150 of file envelope.C.

Referenced by WriteVRMLgr(), EnvelopeConstraint::getGradient(), and gradient().

```
00150                                              {
00151   if ((x < 0) || (x >= xdim) ||
00152       (y < 0) || (y >= ydim) ||
00153       (z < 0) || (z >= zdim))  {
00154 //     cerr << "Envelope const gradient access out of bounds.  I got "
00155 //       << x << ", "<< y << ", "<< z << endl;
00156 //     raise (SIGQUIT);
00157     return JVzero3;
00158   }
00159
```

4.27.2.28 double & Envelope::operator() (int *x*, int *y*, int *z*)

lvalue (for assigning density).

Definition at line 178 of file envelope.C.

```
00178                                          {
00179   if ((x < 0) || (x >= xdim) ||
00180       (y < 0) || (y >= ydim) ||
00181       (z < 0) || (z >= zdim))  {
00182     cerr << "Tried to assign to Envelope out of bounds.  I got "
00183          << x << ", "<< y << ", "<< z << endl;
00184     DIE("");
00185   }
00186
```

4.27.2.29 const double Envelope::operator() (const jVector & *J*) const

access density by position.

Definition at line 203 of file envelope.C.

```
00203                                            {
00204   int x,y,z;
00205
00206   // check this
```

```
00207   x = (int) floor((J[0]-minx)/InputWidth);
00208   y = (int) floor((J[1]-miny)/InputWidth);
00209   z = (int) floor((J[2]-minz)/InputWidth);
00210
00211   return ($\ast$this)(x,y,z);
00212
00213
```

4.27.2.30 const double Envelope::operator() (int *x*, int *y*, int *z*) const

access density by cell.

Definition at line 190 of file envelope.C.

```
00190                                                {
00191   if ((x < 0) || (x >= xdim) ||
00192       (y < 0) || (y >= ydim) ||
00193       (z < 0) || (z >= zdim))  {
00194 //     cerr << "Envelope const access out of bounds.  I got "
00195 //       << x << ", "<< y << ", "<< z << endl;
00196 //     raise (SIGQUIT);
00197     return MIN_ENV_DEN;
00198   }
00199
```

4.27.3 Friends And Related Function Documentation

4.27.3.1 class EnvelopeConstraint [friend]

Definition at line 34 of file envelope.h.

4.27.4 Member Data Documentation

4.27.4.1 double Envelope::InputWidth

Real width of cubes (e.g. 3.7A).

Definition at line 106 of file envelope.h.

4.27.4.2 const jVector Envelope::JVzero3

useful null vector.

Definition at line 180 of file envelope.h.

4.27.4.3 double Envelope::cm_x

Center of mass, weighted by probabilities, (as blocks).

Definition at line 119 of file envelope.h.

4.27.4.4 double Envelope::cm_y

Center of mass, weighted by probabilities, (as blocks).

Definition at line 119 of file envelope.h.

4.27.4.5 double Envelope::cm_z

Center of mass, weighted by probabilities, (as blocks).

Definition at line 119 of file envelope.h.

4.27.4.6 double * Envelope::density_array

Array of doubles storing density.

Definition at line 94 of file envelope.h.

4.27.4.7 int Envelope::filledPoints

Number of non-zero cubes in Envelope.

Definition at line 128 of file envelope.h.

4.27.4.8 jVector * Envelope::gradient_field

Array of jVectors storing gradient.

Definition at line 97 of file envelope.h.

4.27.4.9 double Envelope::maxx

Real position of max edges of region.

Definition at line 112 of file envelope.h.

4.27.4.10 double Envelope::maxy

Real position of max edges of region.

Definition at line 112 of file envelope.h.

4.27.4.11 double Envelope::maxz

Real position of max edges of region.

Definition at line 112 of file envelope.h.

4.27.4.12 double Envelope::minx

Real position of min edges of region.

Definition at line 109 of file envelope.h.

4.27.4.13 double Envelope::miny

Real position of min edges of region.

Definition at line 109 of file envelope.h.

4.27.4.14 double Envelope::minz

Real position of min edges of region.

Definition at line 109 of file envelope.h.

4.27.4.15 Matrix Envelope::my_EigVal

Eigenvalues for the envelope.

Definition at line 131 of file envelope.h.

4.27.4.16 Matrix Envelope::my_EigVec

Eigenvectors for the envelope.

Definition at line 134 of file envelope.h.

4.27.4.17 int Envelope::numPoints

Number of cubes in Envelope.

Definition at line 100 of file envelope.h.

4.27.4.18 double Envelope::r_cm_x

Center of mass, weighted by probabilities, (Real postition!).

Definition at line 122 of file envelope.h.

4.27.4.19 double Envelope::r_cm_y

Center of mass, weighted by probabilities, (Real postition!).

Definition at line 122 of file envelope.h.

4.27.4.20 double Envelope::r_cm_z

Center of mass, weighted by probabilities, (Real postition!).

Definition at line 122 of file envelope.h.

4.27.4.21 double Envelope::totalDensity

Total of all the density numbers in the cubes.

Definition at line 125 of file envelope.h.

4.27.4.22 int Envelope::xdim

Dimensions in number of cubes.

Definition at line 103 of file envelope.h.

4.27.4.23 int Envelope::ydim

Dimensions in number of cubes.

Definition at line 103 of file envelope.h.

4.27.4.24 int Envelope::zdim

Dimensions in number of cubes.

Definition at line 103 of file envelope.h.

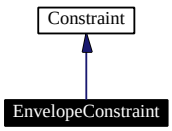
The documentation for this class was generated from the following files:

- **envelope.h**
- **envelope.C**

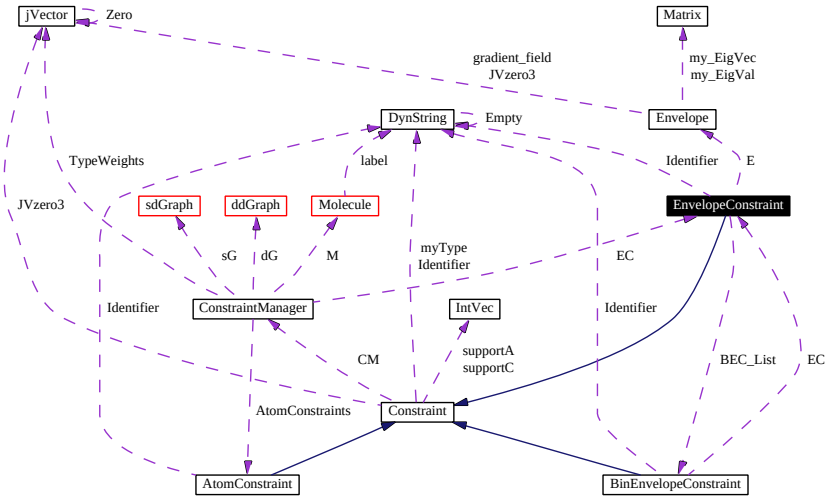
4.28 EnvelopeConstraint Class Reference

`#include <envelope_constraint.h>`

Inheritance diagram for EnvelopeConstraint



Collaboration diagram for EnvelopeConstraint:



Public Methods

- **EnvelopeConstraint** (**ConstraintManager** &cm)

  constructor.

- **~EnvelopeConstraint** ()
- **jVector dStress** (int k, const **jVector** &pos) const
- double **getDensity** (const **jVector** &J) const

  return the density at **jVector** (p. 325) *J.*

- **jVector getGradient** (const **jVector** &J) const

  return the gradient at **jVector** (p. 325) *J.*

- int **Compare** (const EnvelopeConstraint &ec) const
- **MAKECOMPARISONS_WITH_COMPARE** (EnvelopeConstraint)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const
- int **WriteVRML_OrigData** (ostream &outStream) const

  write VRML 1.0 format for initial envelope data (env_A).

- void **MoveModelbyGrad** (double d) const

  move model distance d according to gradient of atoms outsie E[0].

- void **AlignModeltoCOM** () const

  calling this translates & rotates all atoms in the model.

- void **FindEnvelopeEigenvectors** (**Matrix** &EigVec, **Matrix** &EigVal) const

  Determine the EigenVectors for the ""A"" envelope.

- **Matrix FindModelRotation** () const

  Determine the roation matrix.

- void **AlignPrincComp** (void) const

  Rotate current model via pricipal components.

- **jVector FindModelTranslationToCOM** () const

  Determine the translation of model points to COM of envelope.

- double **computeStress** () const

  returns stress on constraint.

- double **computeStress** (int k, const **jVector** &pos) const

  returns stress on constraint if atom k' were in position pos'.

- int **Read** (istream &is)
- int **ReadBinary** (istream &is)

  Same scenerio as **Read***()* (p. 236) *only binary version.*

- int **ReadAvsTxt** (istream &in)

  read text-converted AVS density format.

- void **PreSetup** ()

  aligns the grid.

- void **PostSetup** ()

Public Attributes

- **Envelope**** **E**

  Array of envelope pointers. [0] points to read-in data, rest are blurred copies.

- double **K**

  factor used to combine E[0], E[1], etc. when they overlap.

- **BinEnvelopeConstraint**** **BEC_List**

  array of pointers to BEC's; for access.

Static Public Attributes

- **DynString Identifier** = "envelope:"
- int **num_evelopes** = 2

  Number of envelopes to use. 0 is original data, 1..n-1 are blurred versions.

- double **translation_threshold** = 0.5

  If the translate vector nrm() > this, then a translation occurs.

- double **rotation_threshold** = 0.08

  If the sum(abs(Rotation Matrix)) > this, then a rotation occurs.

- double **density_threshold** = 0.125

  greater than this => atom fits envelope.

- int **blur_kernelSize** = 2

  size of blur kernel.

Friends

- class **BinEnvelopeConstraint**

4.28.1 Constructor & Destructor Documentation

4.28.1.1 EnvelopeConstraint::EnvelopeConstraint (ConstraintManager & *cm*)

constructor.

Definition at line 4 of file envelope_constraint.C.

```
00005   : Constraint(cm,IntVec(cm.numAtoms(),0,1)) // range constructor for intvec
00006   : Constraint(cm)
00007   // NOTE NO SUPPORTS -- will support all the BinEnvConstratints after they're added
00008 {
00009
00010   E = new Envelope*[EnvelopeConstraint::num_evelopes];
00011
00012   E[0] = new Envelope;
00013   for(int i =1;i<EnvelopeConstraint::num_evelopes;i++)
00014     E[i] =0;
00015
00016   BEC_List = 0;
00017   K = 1;
00018
```

4.28.1.2 EnvelopeConstraint::~EnvelopeConstraint ()

Definition at line 22 of file envelope_constraint.C.

```
00022                                       {
00023
00024   for(int i =0;i<EnvelopeConstraint::num_evelopes;i++){
00025     if (E[i])
```

```
00026       delete E[i];
00027   }
00028
```

4.28.2 Member Function Documentation

4.28.2.1 void EnvelopeConstraint::AlignModeltoCOM (void) const

calling this translates & rotates all atoms in the model.

Definition at line 167 of file envelope_constraint.C.

Referenced by computeStress().

```
00167                                                   {
00168
00171
00172   // Translate to COM
00173   jVector T = FindModelTranslationToCOM();
00174   if (T.nrm() > translation_threshold){
00175     CM->TranslateModel(T);
00176 #ifdef NOISY_CONSTRAINTS
00177     cerr << "finished translate by " << T << endl;
00178   } else {
00179     WARN ("not translated -- too close");
00180 #endif
00181   }
00182
```

4.28.2.2 void EnvelopeConstraint::AlignPrincComp (void) const

Rotate current model via pricipal components.

Definition at line 85 of file envelope_constraint.C.

Referenced by AlignModeltoCOM().

```
00085                                                   {
00086
00087   jVector tempV1;
00088   jVector tempV2;
00089   int num = CM->numAtoms();
00090
00091
00092   // Rotate
00093   Matrix Rot(3,3);
00094   Rot = FindModelRotation();
00095
00096   Matrix I(3,3);
00097   I.identity();
00098
00099   if ((Rot - I).SumAbsoluteValue() > rotation_threshold){
00100
00101     for (int a = 0; a < num; a++){
00102       if (CM->constraintON(a)){
00103         for (int i =0;i<3;i++){   //matrix multiply into a jVector
00104
00105           tempV1 = CM->getPosition(a);
00106
00107           //    tempV2[i] =0;
00108           //    for (int j =0;j<3;j++){
```

```
00109           //      tempV2[i] += Rot(i+1,j+1)*tempV1[j];
00110           //    }
00111
00112           tempV2[i]  = Rot(i+1,1)*tempV1[0];
00113           tempV2[i] += Rot(i+1,2)*tempV1[1];
00114           tempV2[i] += Rot(i+1,3)*tempV1[2];
00115
00116         }  // end matrix multiply
00117         CM->setPosition(a, tempV2);
00118       }
00119     } // end loop on num' atoms
00120
00121
00122     //   } else {
00123     // do not move if matrix is almost Identitiy
00124     //     cerr << "Rotate Aborted -- too close to Identity." << endl;
00125
00126   }
00127
```

4.28.2.3 int EnvelopeConstraint::Compare (const EnvelopeConstraint & *ec*) const [inline]

Definition at line 154 of file envelope_constraint.h.

```
00155 {
00156   DIE("Envelope Compare not Implemented");
00157
00158   return 0;
00159
```

4.28.2.4 void EnvelopeConstraint::FindEnvelopeEigenvectors (Matrix & *EigVal*, Matrix & *EigVec*) const

Determine the EigenVectors for the ""A"" envelope.

Definition at line 324 of file envelope_constraint.C.

Referenced by FindModelRotation().

```
00324                                                                                {
00325
00326   E[0]->Get_Eigenvectors(EigVal, EigVec);
00327
00328
```

4.28.2.5 Matrix EnvelopeConstraint::FindModelRotation (void) const

Determine the roation matrix.

Definition at line 352 of file envelope_constraint.C.

Referenced by AlignPrincComp().

```
00352                                              {
00353
00354   Matrix Spread(3,3); //lower triangular matrix of 3x3
00355
```

```
00356   Matrix R; //rotation matrix
00357
00358   jVector tempV;
00359
00360   int in_Points = CM->numAtomsON();
00361
00362   // Calc spread
00363   double dx, dy, dz = 0;
00364   double meanx = 0, meany = 0, meanz = 0;
00365
00366   for (int a = 0; a < in_Points; a++){
00367     if (CM->constraintON(a)){
00368       tempV = CM->getPosition(a);
00369
00370       meanx += tempV[0];
00371       meany += tempV[1];
00372       meanz += tempV[2];
00373     }
00374   }
00375   meanx /= in_Points;
00376   meany /= in_Points;
00377   meanz /= in_Points;
00378
00379   // calc dx dy dz
00380
00381   Spread.zero();
00382
00383
00384   for (int a = 0; a < in_Points; a++){
00385     tempV = CM->getPosition(a);
00386     dx = tempV[0] - meanx;
00387     dy = tempV[1] - meany;
00388     dz = tempV[2] - meanz;
00389
00390     Spread (1,1) += dx*dx/in_Points;
00391     Spread (2,1) += dx*dy/in_Points;
00392     Spread (2,2) += dy*dy/in_Points;
00393     Spread (3,1) += dx*dz/in_Points;
00394     Spread (3,2) += dy*dz/in_Points;
00395     Spread (3,3) += dz*dz/in_Points;
00396   }
00397
```

4.28.2.6 jVector EnvelopeConstraint::FindModelTranslationToCOM (void) const

Determine the translation of model points to COM of envelope.

Definition at line 331 of file envelope_constraint.C.

Referenced by AlignModeltoCOM().

```
00331                                                    {
00332   jVector rv;
00333   jVector model_center = CM->molecule_center();
00334
00335   rv[0] = E[0]->r_cm_x - model_center[0];
00336   rv[1] = E[0]->r_cm_y - model_center[1];
00337   rv[2] = E[0]->r_cm_z - model_center[2];
00338
00339 #ifdef NOISY_CONSTRAINTS
00340   cerr <<"Model COM: "<< model_center << endl;
00341
00342   cerr <<"Env COM: " << E[0]->r_cm_x <<" " << E[0]->r_cm_y <<" "<< E[0]->r_cm_z << endl;
```

```
00343
00344   cerr << "Translation vector: "  << rv << " (mag=" << rv.mag() << ")" <<endl<< endl;
00345 #endif
00346   return rv;
00347
00348
00349
```

4.28.2.7 EnvelopeConstraint::MAKECOMPARISONS_WITH_COMPARE (EnvelopeConstraint)

4.28.2.8 void EnvelopeConstraint::MoveModelbyGrad (double *d*) const

move model distance d according to gradient of atoms outsie E[0].

Definition at line 132 of file envelope_constraint.C.

Referenced by AlignModeltoCOM().

```
00132                                                   {
00133
00136
00137   jVector dir; dir.zero();
00138   // cycle over my BEC, asking them if their atom is outside
00139   // if so, take position of point, determine gradient there,
00140   // and += to dir.
00141   //  move model by d*dir.unit();
00142
00143   jVector pos;
00144   int num = CM->numAtoms(); // ASSUMES all atoms in CM are in env. <?>
00145   for (int i = 0; i < num; i++){
00146     if (CM->constraintON(i)){
00147       //    cerr << "atom = " << i << endl;
00148       //      pos = CM->getPosition(BEC_List[i]->Atom_id);
00149       pos = CM->getPosition(i);
00150       if ((*E[0])(pos) <= EnvelopeConstraint::density_threshold) {// outside orig data
00151         //      cerr << "location = " << pos << endl;
00152         //      cerr << "gradient = " << getGradient(pos) << endl;
00153         dir += getGradient(pos);
00154       }
00155     }
00156   }
00157
```

4.28.2.9 int EnvelopeConstraint::Output (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 439 of file envelope_constraint.C.

```
00440 {
00441   /*
00442     os << supportA[0] << ' ' << supportA[1] << ' '
00443     << measured << ' ' << variance;
00444     return 2;
00445   */
00446   return 0;
00447
```

4.28.2.10 int EnvelopeConstraint::OutputBinary (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 458 of file envelope_constraint.C.

```
00459 {
00460   //  os <= supportA <= measured <= variance;
00461   return 2;
00462
```

4.28.2.11 void EnvelopeConstraint::PostSetup (void) [virtual]

constraints get their ID when added to the constraint graph. this addition is done in the Input function between the pre- and post setup functions. constraint setup that needs to occur with an ID and an added constraint go in post setup. (like adding parent constraints, etc.)

Reimplemented from **Constraint** (p. 84).

Definition at line 57 of file envelope_constraint.C.

```
00057                                   {
00058
00059   BinEnvelopeConstraint * BECptr;
00060
00061   int num = CM->numAtoms(); // ASSUMES all atoms in CM are in env. <?> FIX
00062   BEC_List = new BinEnvelopeConstraint * [num];
00063
00064   for (int i = 0; i < num; i++){
00065
00066     BECptr = new BinEnvelopeConstraint(*CM,i,this);
00067     // to be deleted by ~ConstraintManager
00068
00069     BEC_List[i] = BECptr; // record it.
00070
00071     BECptr->Input(cin); // INPUT STREAM NOT USED (is there a null stream?)
00072
00073     // each bin_envelope is supported by the envelope_constraint
00074     // When the envelope_constraint is "valid" the atoms have been
00075     // aligned ok against the envelope.  Asserting the highlevel EnvelopeConstraint
00076     // invalid will force further alignment
00077
00078   }
00079
```

4.28.2.12 void EnvelopeConstraint::PreSetup (void) [virtual]

aligns the grid.

Reimplemented from **Constraint** (p. 85).

Definition at line 36 of file envelope_constraint.C.

```
00036                                   {
00037
00038   // create blurred envelopes
00039
00040   if (E[0]->numPoints > 1) {
00041
00042     for(int i =1;i<EnvelopeConstraint::num_evelopes;i++){
```

```
00043       E[i] = new Envelope(*E[i-1]); // copy
00044       E[i]->Blur(1);                // FIX -- make a variable in globals
00045       E[i]->Upsize(2);
00046     }
00047
00048   } else {
00049     cerr << " Envelope setup failed.  Envelope has " << E[0]->numPoints << " points." << endl;
00050     DIE("");
00051   }
00052
```

4.28.2.13 int EnvelopeConstraint::Read (istream & *is*) [virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 236) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 236) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented from **Constraint** (p. 85).

Definition at line 449 of file envelope_constraint.C.

```
00450 {
00451   // tell A to read away!
00452   int rv = E[0]->Read(is);
00453   E[0]->Center();
00454   return rv;
00455
00456
```

4.28.2.14 int EnvelopeConstraint::ReadAvsTxt (istream & *in*)

read text-converted AVS density format.

4.28.2.15 int EnvelopeConstraint::ReadBinary (istream & *is*) [virtual]

Same scenerio as **Read**() (p. 236) only binary version.

Reimplemented from **Constraint** (p. 85).

Definition at line 464 of file envelope_constraint.C.

```
00465 {
00466   //  is >= supportA[0] >= supportA[1] >= measured >= variance;
00467   return 2;
00468
```

4.28.2.16 int EnvelopeConstraint::WriteVRML_OrigData (ostream & *outStream*) const [inline]

write VRML 1.0 format for initial envelope data (env_A).

Definition at line 162 of file envelope_constraint.h.

Referenced by jdErr_Eval::run().

```
00162                                                                    {
00163   return E[0]->WriteVRML(outStream);
00164
```

4.28.2.17 double EnvelopeConstraint::computeStress (int *k*, const jVector & *pos_k*) const [virtual]

returns stress on constraint if atom k' were in position pos'.

Reimplemented from **Constraint** (p. 89).

Definition at line 225 of file envelope_constraint.C.

```
00226 {
00227
00228   DIE("");
00229     return 0;
00230
00231
00232
```

4.28.2.18 double EnvelopeConstraint::computeStress () const [virtual]

returns stress on constraint.

Reimplemented from **Constraint** (p. 89).

Definition at line 210 of file envelope_constraint.C.

```
00210                                       {
00211
00212   // align centroid
00213
00214   // calc rotation matirix
00215
00216   // rotate atoms
00217
00218   AlignModeltoCOM();
00219   return 0;
00220
00221
```

4.28.2.19 jVector EnvelopeConstraint::dStress (int *k*, const jVector & *pos_k*) const [virtual]

asking constraint number k for its 12 element dStress vector we pass in pos_a vector – corresponding to the atom position if the constraint is an atom, and corresponding to the position of the atom k in question if the call is to a non-atom constraint. EG for a distance constraint atom 12 needs the dStress from the d-constraint, so it will call dStress on the d-constraint(12, atom_pos) so the d_constr can answer the question if atom 12 were in atom_pos, I would say my dStress is blah.

Reimplemented from **Constraint** (p. 89).

Definition at line 235 of file envelope_constraint.C.

```
00236 {
00237
00238   //  DIE("");
```

```
00239   return JVzero3;
00240
00241
```

4.28.2.20 double EnvelopeConstraint::getDensity (const jVector & *J*) const

return the density at **jVector** (p. 325) J.

Definition at line 244 of file envelope_constraint.C.

Referenced by BinEnvelopeConstraint::computeStress(), and BinEnvelopeConstraint::dStress().

```
00244                                                          {
00245
00246
00247   double D = (*E[0])(J) ;
00248   //  cerr << D << " = E[0] density \n" ;
00249   if (D >  EnvelopeConstraint::density_threshold)
00250     return MAX_ENV_DEN;
00251
00252   int i = 1;
00253   while ((i < EnvelopeConstraint::num_evelopes) && (D < EnvelopeConstraint::density_threshold))
00254     {
00255       D = (*E[i])(J) ;
00256       //      cerr << "next, " << D << " = E[" <<i<<"] density \n" ;
00257       i++;
00258
00259     }
00260
00261 // else {
00262 //   //  double rv = (1/(1+K))*(((*E[0])(J))+(K*((*env_B)(J))));
00263 //     rv = util_max( a_density , K*(*env_B)(J) );
00264 //     }
00265 //   // DEBUG
00266 // //   cerr << endl << "EC::getDensity: pos{" << J[0] << ", " << J[1]
00267 // //        << ", " << J[2] << "}
00268
```

4.28.2.21 jVector EnvelopeConstraint::getGradient (const jVector & *J*) const

return the gradient at **jVector** (p. 325) J.

Definition at line 274 of file envelope_constraint.C.

Referenced by MoveModelbyGrad(), and BinEnvelopeConstraint::dStress().

```
00274                                                              {
00275
00276   double D = (*E[0])(J);
00277   jVector G = E[0]->gradient(J) ;
00278   //  cerr << D << " = E[0] density " ;
00279   if (D >  EnvelopeConstraint::density_threshold)
00280     return G;
00281
00282   int i = 1;
00283   while ((i < EnvelopeConstraint::num_evelopes) && (D < EnvelopeConstraint::density_threshold))
00284     {
00285       D = (*E[i])(J) ;
00286       G = E[i]->gradient(J);
00287       //  cerr << D << " = E[" <<i<<"] density " ;
00288       i++;
```

```
00289     }
00290
00291    //
00292 //   jVector G;
00293 //   double p = (*E[0])(J);
00294 //   if ( p >= EnvelopeConstraint::density_threshold ) {
00295 //     if ( p >= K*(*env_B)(J) ) {
00296 //       //    cerr << p <<"(A),  ";
00297 //       G = E[0]->gradient(J);
00298 //     }  else {
00299 //       G = env_B->gradient(J);
00300 //     }
00301 //   } else {
00302 //     G = JVzero;
00303 //   }
00304
```

4.28.3 Friends And Related Function Documentation

4.28.3.1 class BinEnvelopeConstraint [friend]

Definition at line 32 of file envelope_constraint.h.

Referenced by PostSetup().

4.28.4 Member Data Documentation

4.28.4.1 BinEnvelopeConstraint ** EnvelopeConstraint::BEC_List

array of pointers to BEC's; for access.

Definition at line 75 of file envelope_constraint.h.

4.28.4.2 Envelope ** EnvelopeConstraint::E

Array of envelope pointers. [0] points to read-in data, rest are blurred copies.

Definition at line 69 of file envelope_constraint.h.

4.28.4.3 DynString EnvelopeConstraint::Identifier = "envelope:" [static]

Reimplemented from **Constraint** (p. 95).

Definition at line 105 of file globals.C.

4.28.4.4 double EnvelopeConstraint::K

factor used to combine E[0], E[1], etc. when they overlap.

Definition at line 72 of file envelope_constraint.h.

4.28.4.5 int EnvelopeConstraint::blur_kernelSize = 2 [static]

size of blur kernel.

Definition at line 110 of file globals.C.

4.28.4.6 double EnvelopeConstraint::density_threshold = 0.125 [static]

greater than this => atom fits envelope.

Definition at line 107 of file globals.C.

4.28.4.7 int EnvelopeConstraint::num_evelopes = 2 [static]

Number of envelopes to use. 0 is original data, 1..n-1 are blurred versions.

Definition at line 106 of file globals.C.

4.28.4.8 double EnvelopeConstraint::rotation_threshold = 0.08 [static]

If the sum(abs(Rotation Matrix)) > this, then a rotation occurs.

Definition at line 108 of file globals.C.

4.28.4.9 double EnvelopeConstraint::translation_threshold = 0.5 [static]

If the translate vector nrm() > this, then a translation occurs.

Definition at line 109 of file globals.C.

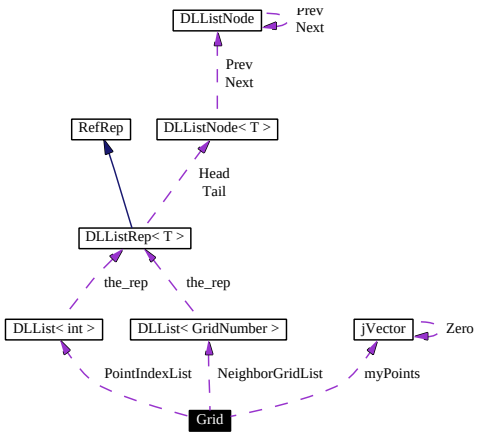
The documentation for this class was generated from the following files:

- **envelope_constraint.h**
- **envelope_constraint.C**
- **globals.C**

4.29 Grid Class Reference

`#include <grid.h>`

Collaboration diagram for Grid:



Public Methods

- **Grid** ()
- **Grid** (**pointList** plist, int numP, double **delta**, double **Length**=0)
- **Grid** (int in_numPoints, double in_delta, double in_Length)
- **Grid** (const Grid &G)
- **~Grid** ()
- double **getBounds** (void) const
- double **getDelta** (void) const
- int **getNumPoints** (void) const
- const **pointList getPoints** (void) const
- const **GridNumber*** **getGrids** (void) const
- **GridNumber getGrid** (int point_index) const
- **GridNumber getGridDimension** () const
- double **getDistance** (int point1_index, int point2_index) const
- double **getDistance** (int point1_index, const **jVector** &p) const
- double **getDistance** (const **jVector** &p, int point2_index) const
- const **jVector**& **getPosition** (int point_index) const
- bool **JVoverlap** (const **jVector** &point) const
- bool **GNoverlap** (**GridNumber** GN) const
- **DLList**<double> **neighbor_d** (int index, double distance) const
- **DLList**<double> **neighbor_d** (const **jVector** &p, double distance) const
- **DLList**<**jVector**> **neighbor_p** (int index, double distance) const
- **DLList**<**jVector**> **neighbor_p** (const **jVector** &p, double distance) const
- **DLList**<**jVector**> **approx_neighbor_p** (int index, double distance) const
- **DLList**<**jVector**> **approx_neighbor_p** (const **jVector** &p, double distance) const
- **DLList**<int> **neighbor_i** (int index, double distance) const
- **DLList**<int> **neighbor_i** (const **jVector** &p, double distance) const
- **DLList**<int> **approx_neighbor_i** (int index, double distance) const
- **DLList**<int> **approx_neighbor_i** (const **jVector** &p, double distance) const

- double **min_neighbor_d** (const **jVector** &p, double distance) const
- void **ReGrid** (**pointList** plist, int numP, double in_delta, double in_Length=0)
- void **ReSize** (int numP, double in_delta, double in_Length)
- void **setPosition** (int index_num, const **jVector** &NewPosition)
- int **addPoints** (**pointList** pList, int numP)
- int **addPoint** (const **jVector** &Position)
- int **deletePoint** (int index_num)
- void **clear** (void)

Private Methods

- void **Initialize** (**pointList** pList, int numP)
- void **setupNeighborLists** ()
- double **calcDistance3** (const **jVector** &j1, const **jVector** &j2) const
- **GridNumber createHash** (const **jVector** &P) const

Private Attributes

- double **Length**
- double **TwiceLength**
- double **N1Dist**
- double **N2Dist**
- double **delta**
- int **numPoints**
- **pointList myPoints**
- **hashmap_type GridHashMap**
- **DLList<GridNumber> NeighborGridList**
- **DLList<int> PointIndexList**
- **GridNumber* PointToGridNumArray**
- **GridNumber GridDimension**
- **GridNumber GridDimensionSq**
- hashmap_type::iterator** **PointToHashIter**
- **GridNumber Neighbor1** [27]
- **GridNumber Neighbor2** [125]

4.29.1 Constructor & Destructor Documentation

4.29.1.1 Grid::Grid ()

Definition at line 9 of file grid.C.

```
00009           {
00010     Length = 0;
00011     delta = 0.0;
00012     numPoints = 0;
00013
```

4.29.1.2 Grid::Grid (pointList *plist*, int *in_numPoints*, double *in_delta*, double *in_Length* = 0)

Definition at line 16 of file grid.C.

```
00017 {
00018
00019   numPoints = in_numPoints;
00020   delta = in_delta;
00021   Length = in_Length;
00022   Initialize(plist,numPoints);
00023
```

4.29.1.3 Grid::Grid (int *in_numPoints*, double *in_delta*, double *in_Length*)

Definition at line 34 of file grid.C.

```
00035 {
00036   jVector * tempList = new jVector [in_numPoints];
00037
00038   // create dummy starting posistions
00039
00040   for (int i=0; i<in_numPoints;i++){
00041     tempList[i].Resize(3);
00042     tempList[i].zero();
00043     //    tempPos = ((((.5)*i)/in_numPoints)+(0.25))*in_Length;
00044   }
00045
```

4.29.1.4 Grid::Grid (const Grid & *G*)

Definition at line 25 of file grid.C.

```
00025                       {
00026
00027   numPoints = G.getNumPoints();
00028   delta = G.getDelta();
00029   Length = G.getBounds();
00030   Initialize(G.getPoints(),numPoints);
00031
00032
```

4.29.1.5 Grid::~Grid ()

Definition at line 56 of file grid.C.

```
00057 {
00058   clear();
00059
```

4.29.2 Member Function Documentation

4.29.2.1 bool Grid::GNoverlap (GridNumber *GN*) const

Definition at line 189 of file grid.C.
Referenced by JVoverlap().

```
00189                                       {
00190
00191   pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00192   iter_pair = GridHashMap.equal_range(GN);
00193   if (iter_pair.first != iter_pair.second){
00194     // an atom exits in this grid
00195     return true;
00196   }
00197
```

4.29.2.2 void Grid::Initialize (pointList *plist*, int *numP*) [private]

Definition at line 61 of file grid.C.

Referenced by Grid(), and ReGrid().

```
00061                                                 {
00062
00063   // Reset bounding box width (length') if too small
00064   double temp;
00065   double tempLength  = 0;
00066   for(int i=0; i<numP; i++) {
00067     for(int j=0; j<3; j++) {
00068       //      if (plist[i]){ //sanity check-- make sure a jVecotr is there LEAKING
00069         temp = fabs(plist[i][j]);
00070         if (tempLength < temp ) tempLength = temp;
00071         //      }
00072     }
00073   }
00074   if (tempLength >= Length) {
00075     // minimum bounds condition
00076     Length = tempLength;
00077   }
00078
00079   numPoints = numP; // different if resze or regrid is called.
00080   TwiceLength = 2 * Length;  //Commonly reused multiples, calculated once.
00081   N1Dist = delta;
00082   N2Dist = 2 * delta;
00083
00084   GridDimension = (GridNumber) ((TwiceLength+delta)/delta);
00085
00086   if (GridDimension > 1290) { // fix to check actual limit == (max long int) ^-3
00087     cerr << "You have attempted to create a Grid with more" << endl
00088          << "than 1290 elements in each dimension.  This is" << endl
00089          << "not allowed because of the limitations on the" << endl
00090          << "of long integers used on this system." << endl;
00091     cerr << "The number of elements you tried to use was "
00092          << GridDimension << " ." << endl << endl;
00093     exit(1);
00094   }
00095
00096   GridDimensionSq =  GridDimension*GridDimension;
00097
00098   setupNeighborLists();
00099
00100   GridHashMap.clear();
00101   GridHashMap.resize(5*numP);  // MAGIC NUMBER (use this many hash buckets)
00102
00103   PointToGridNumArray = new GridNumber[numP];
00104   myPoints = new jVector[numP];
00105   PointToHashIter = new hashmap_type::iterator * [numP];
00106   GridNumber TempGridNumber;
00107
```

```
00108   // Go through all the input points
00109   for(int i=0; i<numP; i++) {
00110     // Save a copy of the location vector
00111     myPoints[i] = plist[i];
00112
00113     // Create a long int to hash the filled grids
00114     TempGridNumber = createHash(myPoints[i]);
00115
00116     // Add elements to the GridHashMap
00117     PointToHashIter[i] = new hashmap_type::iterator;
00118     *PointToHashIter[i] = GridHashMap.insert(hashmap_type::value_type(TempGridNumber, i));
00119
00120     // Save the hash number for the grid
00121     PointToGridNumArray[i] = TempGridNumber;
00122   }
00123
```

4.29.2.3 bool Grid::JVoverlap (const jVector & *point*) const

Definition at line 201 of file grid.C.

```
00201                                           {
00202
00203   GridNumber boxNumber = createHash(point);
00204   return GNoverlap(boxNumber);
00205
00206
```

4.29.2.4 void Grid::ReGrid (pointList *plist*, int *numP*, double *in_delta*, double *in_Length* = 0)

Definition at line 158 of file grid.C.

Referenced by ReSize().

```
00158                                                                                {
00159   // RE-allocate new Grid box containing all points in the list
00160   //  This should be  called if a point is moved outside the current
00161   //  bounds
00162
00163   clear();
00164
00165   delta = in_delta;
00166   Length = in_Length;
00167   Initialize(plist, numP);
00168
00169
00170
```

4.29.2.5 void Grid::ReSize (int *numP*, double *in_delta*, double *in_Length*)

Definition at line 172 of file grid.C.

```
00172                                                       {
00173
00174   jVector * tempList = new jVector [numP];
00175
00176   // create dummy starting posistions
```

```
00177
00178   for (int i=0; i<numP;i++){
00179     tempList[i].Resize(3);
00180     tempList[i].zero();
00181   }
00182
```

4.29.2.6 int Grid::addPoint (const jVector & *Position*) [inline]

Definition at line 150 of file grid.h.

```
00150                                       {/*not implemented yet*/return 0;}
00151
```

4.29.2.7 int Grid::addPoints (pointList *pList*, int *numP*) [inline]

Definition at line 142 of file grid.h.

```
00142                                         {/*Not yet*/ return 0;}
00143
```

4.29.2.8 DLList< int > Grid::approx_neighbor_i (const jVector & *p*, double *distance*) const

Definition at line 683 of file grid.C.

```
00683                                                                        {
00684 // returns a doubleley-linked list of point **indicies** within all the
00685 // grids that are 'distance' away from point p.
00686 // All points in qualifying grids are returned.  This means it is a
00687 // correct list, yet may contain extra elements.
00688   DLList<int> ReturnList;
00689   if (distance <= N1Dist) {
00690     // Go through 27 one-neighbor grids and add up contributions
00691     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00692     hashmap_type::const_iterator out_iter;
00693     GridNumber TempGridNumber = createHash(p);
00694     for (int i = 0; i < 27; i++){
00695       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00696       if (iter_pair.first != iter_pair.second){
00697         // an atom exits in this grid
00698         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00699           ReturnList.AddToFront((*out_iter).second); // index
00700         }
00701       }
00702     } // end of neighbor1 cycles
00703   } else if (distance <= N2Dist){
00704     // Go through 27 one-neighbor grids and add up contributions
00705     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00706     hashmap_type::const_iterator out_iter;
00707     GridNumber TempGridNumber = createHash(p);
00708     for (int i = 0; i < 125; i++){
00709       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00710       if (iter_pair.first != iter_pair.second){
00711         // an atom exits in this grid
00712         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00713           ReturnList.AddToFront((*out_iter).second); // index
```

```
00714         }
00715       }
00716     } // end of neighbor2 cycles
00717   } else { // outside near neighbors, return all
00718     for (int i = 0; i < numPoints; i++){
00719       ReturnList.AddToFront(i);                     // index
00720     }
00721   }
00722
```

4.29.2.9 DLList< int > Grid::approx_neighbor_i (int *index*, double *distance*) const

Definition at line 633 of file grid.C.

Referenced by Molecule::approx_neighbor_i().

```
00633                                                                    {
00634   jVector p = myPoints[index];
00635 // returns a doublely-linked list of point **indicies** within all the
00636 // grids that are 'distance' away from point p.
00637 // All points in qualifying grids are returned.  This means it is a
00638 // correct list, yet may contain extra elements.
00639   DLList<int> ReturnList;
00640   if (distance <= N1Dist) {
00641     // Go through 27 one-neighbor grids and add up contributions
00642     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00643     hashmap_type::const_iterator out_iter;
00644     GridNumber TempGridNumber = createHash(p);
00645     for (int i = 0; i < 27; i++){
00646       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00647       if (iter_pair.first != iter_pair.second){
00648         // an atom exits in this grid
00649         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00650           if (out_iter != *(PointToHashIter[index])){
00651             ReturnList.AddToFront((*out_iter).second); // index
00652           }
00653         }
00654       }
00655     } // end of neighbor1 cycles
00656   } else if (distance <= N2Dist){
00657     // Go through 27 one-neighbor grids and add up contributions
00658     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00659     hashmap_type::const_iterator out_iter;
00660     GridNumber TempGridNumber = createHash(p);
00661     for (int i = 0; i < 125; i++){
00662       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00663       if (iter_pair.first != iter_pair.second){
00664         // an atom exits in this grid
00665         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00666           if (out_iter != *(PointToHashIter[index])){
00667             ReturnList.AddToFront((*out_iter).second); // index
00668           }
00669         }
00670       }
00671     } // end of neighbor2 cycles
00672   } else { // outside near neighbors, return all
00673     for (int i = 0; i < numPoints; i++){
00674       if (i != index ) {
00675         ReturnList.AddToFront(i);                     // index
00676       }
00677     }
00678   }
00679
```

4.29.2.10 DLList< jVector > Grid::approx_neighbor_p (const jVector & *p*, double *distance*) const

Definition at line 491 of file grid.C.

```
00491                                                                    {
00492 // Returnsa an approximate list of **positions** to other points within
00493 // a given distance.  This is approximate because the distances are
00494 // not calculated here.  All points in qualifying grids are
00495 // returned.  This means it is a correct list, yet may contain extra
00496 // elements.
00497
00498   DLList<jVector> ReturnList;
00499   if (distance <= N1Dist) {
00500     // Go through 27 one-neighbor grids and add up contributions
00501     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00502     hashmap_type::const_iterator out_iter;
00503     GridNumber TempGridNumber = createHash(p);
00504     for (int i = 0; i < 27; i++){
00505       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00506       if (iter_pair.first != iter_pair.second){
00507         // an atom exits in this grid
00508         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00509           ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00510         }
00511       }
00512     } // end of neighbor1 cycles
00513   } else if (distance <= N2Dist){
00514     // Go through 27 one-neighbor grids and add up contributions
00515     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00516     hashmap_type::const_iterator out_iter;
00517     GridNumber TempGridNumber = createHash(p);
00518     for (int i = 0; i < 125; i++){
00519       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00520       if (iter_pair.first != iter_pair.second){
00521         // an atom exits in this grid
00522         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00523           ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00524         }
00525       }
00526     } // end of neighbor2 cycles
00527   } else { // outside near neighbors, return all
00528     for (int i = 0; i < numPoints; i++){
00529       ReturnList.AddToFront(myPoints[i]); // jVector position
00530     }
00531   }
00532
```

4.29.2.11 DLList< jVector > Grid::approx_neighbor_p (int *index*, double *distance*) const

Definition at line 438 of file grid.C.

```
00438                                                                  {
00439   jVector p = myPoints[index];
00440 // Returnsa an approximate list of **positions** to other points within
00441 // a given distance.  This is approximate because the distances are
00442 // not calculated here.  All points in qualifying grids are
00443 // returned.  This means it is a correct list, yet may contain extra
00444 // elements.
00445
00446   DLList<jVector> ReturnList;
```

```
00447   if (distance <= N1Dist) {
00448     // Go through 27 one-neighbor grids and add up contributions
00449     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00450     hashmap_type::const_iterator out_iter;
00451     GridNumber TempGridNumber = createHash(p);
00452     for (int i = 0; i < 27; i++){
00453       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00454       if (iter_pair.first != iter_pair.second){
00455         // an atom exits in this grid
00456         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00457           if (out_iter != *(PointToHashIter[index])){
00458             ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00459           }
00460         }
00461       }
00462     } // end of neighbor1 cycles
00463   } else if (distance <= N2Dist){
00464     // Go through 27 one-neighbor grids and add up contributions
00465     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00466     hashmap_type::const_iterator out_iter;
00467     GridNumber TempGridNumber = createHash(p);
00468     for (int i = 0; i < 125; i++){
00469       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00470       if (iter_pair.first != iter_pair.second){
00471         // an atom exits in this grid
00472         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00473           if (out_iter != *(PointToHashIter[index])){
00474             ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00475           }
00476         }
00477       }
00478     } // end of neighbor2 cycles
00479   } else { // outside near neighbors, return all
00480     for (int i = 0; i < numPoints; i++){
00481       if (i != index){
00482         ReturnList.AddToFront(myPoints[i]); // jVector position
00483       }
00484     }
00485   }
00486
```

4.29.2.12 double Grid::calcDistance3 (const jVector & *j1*, const jVector & *j2*) const [inline, private]

Definition at line 247 of file grid.h.

Referenced by getDistance().

```
00247                                                                    {
00248
```

4.29.2.13 void Grid::clear (void)

Definition at line 728 of file grid.C.

Referenced by ReGrid(), and ~Grid().

```
00729 {
00730   // Empties the Grid, deletes and zeros all dynamic memory.
00731
```

```
00732   GridHashMap.clear();
00733
00734   for (int i = 0; i< numPoints; i++){
00735     delete PointToHashIter[i];
00736   }
00737
```

4.29.2.14 GridNumber Grid::createHash (const jVector & *P*) const [inline, private]

Definition at line 223 of file grid.h.

Referenced by Initialize(), JVoverlap(), approx_neighbor_i(), approx_neighbor_p(), neighbor_d(), neighbor_i(), neighbor_p(), and setPosition().

```
00223                                                    {
00224     GridNumber Hashx, Hashy, Hashz;
00225     Hashx = (GridNumber) (((P[0] + Length)/TwiceLength)*(GridDimension-1));
00226     Hashy = (GridNumber) (((P[1] + Length)/TwiceLength)*(GridDimension-1));
00227     Hashz = (GridNumber) (((P[2] + Length)/TwiceLength)*(GridDimension-1));
00228     return Hashx + GridDimension * Hashy + GridDimensionSq * Hashz;
00229
```

4.29.2.15 int Grid::deletePoint (int *index_num*) [inline]

Definition at line 156 of file grid.h.

```
00156                                  {/*not implemented yet*/return 0;}
00157
```

4.29.2.16 double Grid::getBounds (void) const [inline]

Definition at line 261 of file grid.h.

Referenced by Grid().

```
00262     { return Length; }
00263
```

4.29.2.17 double Grid::getDelta (void) const [inline]

Definition at line 264 of file grid.h.

Referenced by Grid().

```
00265   { return delta; }
00266
```

4.29.2.18 double Grid::getDistance (const jVector & *p*, int *point2_index*) const [inline]

Definition at line 257 of file grid.h.

```
00257                                                                {
00258
```

4.29.2.19 double Grid::getDistance (int *point1_index*, const jVector & *p*) const [inline]

Definition at line 254 of file grid.h.

```
00254                                                                 {
00255
```

4.29.2.20 double Grid::getDistance (int *point1_index*, int *point2_index*) const [inline]

Definition at line 72 of file grid.h.

Referenced by neighbor_d(), neighbor_i(), and neighbor_p().

```
00072                                                         {
00073
```

4.29.2.21 GridNumber Grid::getGrid (int *point_index*) const [inline]

Definition at line 276 of file grid.h.

```
00276                                                 {
00277     return PointToGridNumArray[point_index];
00278
```

4.29.2.22 GridNumber Grid::getGridDimension () const [inline]

Definition at line 281 of file grid.h.

```
00281                                         {
00282     return GridDimension;
00283
```

4.29.2.23 const GridNumber * Grid::getGrids (void) const [inline]

Definition at line 273 of file grid.h.

```
00274   { return PointToGridNumArray; }
00275
```

4.29.2.24 int Grid::getNumPoints (void) const [inline]

Definition at line 267 of file grid.h.

Referenced by Grid().

```
00268   { return numPoints; }
00269
```

4.29.2.25 const pointList Grid::getPoints (void) const [inline]

Definition at line 270 of file grid.h.

Referenced by Grid().

```
00271   { return myPoints; }
00272
```

4.29.2.26 const jVector & Grid::getPosition (int *point_index*) const [inline]

Definition at line 250 of file grid.h.

Referenced by Molecule::Bounds(), Molecule::Copy(), Molecule::getCOM(), Molecule::getPosition(), Molecule::operator()(), Molecule::writeXmol(), and Molecule::writeXmolData().

```
00250                                                    {
00251
```

4.29.2.27 double Grid::min_neighbor_d (const jVector & *p*, double *distance*) const

Definition at line 726 of file grid.C.

```
00726                                                        {/*not yet*/return 0.0;}
00727
```

4.29.2.28 DLList< double > Grid::neighbor_d (const jVector & *p*, double *distance*) const

Definition at line 298 of file grid.C.

```
00298                                                          {
00299 // Returns a list of **distances** to other points within a given
00300 // distance to the first point.
00301   DLList<double> ReturnList;
00302   double d;
00303   if (distance <= N1Dist) {
00304     // Go through 27 one-neighbor grids and add up contributions
00305     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00306     hashmap_type::const_iterator out_iter;
00307     GridNumber TempGridNumber = createHash(p);
00308     for (int i = 0; i < 27; i++){
00309       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00310       if (iter_pair.first != iter_pair.second){
00311         // an atom exits in this grid
00312         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00313           d = getDistance((*out_iter).second,p); // distance
00314           if (d < distance) ReturnList.AddToFront(d);
00315         }
00316       }
00317     } // end of neighbor1 cycles
00318   } else if (distance <= N2Dist){
00319     // Go through 27 one-neighbor grids and add up contributions
00320     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00321     hashmap_type::const_iterator out_iter;
00322     GridNumber TempGridNumber = createHash(p);
00323     for (int i = 0; i < 125; i++){
```

```
00324       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00325       if (iter_pair.first != iter_pair.second){
00326         // an atom exits in this grid
00327         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00328           d = getDistance((*out_iter).second,p); // distance
00329           if (d < distance) ReturnList.AddToFront(d);
00330         }
00331       }
00332     } // end of neighbor2 cycles
00333   } else { // outside near neighbors, return all
00334     for (int i = 0; i < numPoints; i++){
00335       d = getDistance(i,p); // distance
00336       if (d < distance) ReturnList.AddToFront(d);
00337     }
00338   }
00339
```

4.29.2.29 DLList< double > Grid::neighbor_d (int *index*, double *distance*) const

Definition at line 246 of file grid.C.

```
00246                                                                    {
00247   jVector p = myPoints[index];
00248 // Returns a list of **distances** to other points within a given
00249 // distance to the first point.
00250   DLList<double> ReturnList;
00251   double d;
00252   if (distance <= N1Dist) {
00253     // Go through 27 one-neighbor grids and add up contributions
00254     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00255     hashmap_type::const_iterator out_iter;
00256     GridNumber TempGridNumber = createHash(p);
00257     for (int i = 0; i < 27; i++){
00258       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00259       if (iter_pair.first != iter_pair.second){
00260         // an atom exits in this grid
00261         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00262           if (out_iter != *(PointToHashIter[index])){
00263             d = getDistance((*out_iter).second,p); // distance
00264             if (d < distance) ReturnList.AddToFront(d);
00265           }
00266         }
00267       }
00268     } // end of neighbor1 cycles
00269   } else if (distance <= N2Dist){
00270     // Go through 27 one-neighbor grids and add up contributions
00271     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00272     hashmap_type::const_iterator out_iter;
00273     GridNumber TempGridNumber = createHash(p);
00274     for (int i = 0; i < 125; i++){
00275       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00276       if (iter_pair.first != iter_pair.second){
00277         // an atom exits in this grid
00278         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00279           if (out_iter != *(PointToHashIter[index])){
00280             d = getDistance((*out_iter).second,p); // distance
00281             if (d < distance) ReturnList.AddToFront(d);
00282           }
00283         }
00284       }
00285     } // end of neighbor2 cycles
00286   } else { // outside near neighbors, return all
00287     for (int i = 0; i < numPoints; i++){
```

```
00288       if (i != index){
00289         d = getDistance(i,p); // distance
00290         if (d < distance) ReturnList.AddToFront(d);
00291       }
00292     }
00293   }
00294
```

4.29.2.30 DLList< int > Grid::neighbor_i (const jVector & *p*, double *distance*) const

Definition at line 588 of file grid.C.

```
00588                                                                    {
00589   // returns a doubly-linked list of point **indicies** within all the
00590   // grids that are 'distance' away from point p. NB this calclulates
00591   // and discards distances. (SLOW)
00592   DLList<int> ReturnList;
00593   double d;
00594   if (distance <= N1Dist) {
00595     // Go through 27 one-neighbor grids and add up contributions
00596     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00597     hashmap_type::const_iterator out_iter;
00598     GridNumber TempGridNumber = createHash(p);
00599     for (int i = 0; i < 27; i++){
00600       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00601       if (iter_pair.first != iter_pair.second){
00602         // an atom exits in this grid
00603         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00604           d = getDistance((*out_iter).second,p); // distance
00605           if (d< distance) ReturnList.AddToFront((*out_iter).second); // index
00606         }
00607       }
00608     } // end of neighbor1 cycles
00609   } else if (distance <= N2Dist){
00610     // Go through 27 one-neighbor grids and add up contributions
00611     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00612     hashmap_type::const_iterator out_iter;
00613     GridNumber TempGridNumber = createHash(p);
00614     for (int i = 0; i < 125; i++){
00615       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00616       if (iter_pair.first != iter_pair.second){
00617         // an atom exits in this grid
00618         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00619           d = getDistance((*out_iter).second,p); // distance
00620           if (d< distance) ReturnList.AddToFront((*out_iter).second); // index
00621         }
00622       }
00623     } // end of neighbor2 cycles
00624   } else { // outside near neighbors, return all
00625     for (int i = 0; i < numPoints; i++){
00626       d = getDistance(i,p); // distance
00627       if (d < distance) ReturnList.AddToFront(i); // index
00628     }
00629   }
00630
```

4.29.2.31 DLList< int > Grid::neighbor_i (int *index*, double *distance*) const

Definition at line 535 of file grid.C.

Referenced by Molecule::neighbor_i().

```
00535                                                          {
00536   jVector p = myPoints[index];
00537   // returns a doublely-linked list of point **indicies** within all the
00538   // grids that are 'distance' away from point p. NB this calclulates
00539   // and discards distances. (SLOW)
00540   DLList<int> ReturnList;
00541   double d;
00542   if (distance <= N1Dist) {
00543     // Go through 27 one-neighbor grids and add up contributions
00544     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00545     hashmap_type::const_iterator out_iter;
00546     GridNumber TempGridNumber = createHash(p);
00547     for (int i = 0; i < 27; i++){
00548       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00549       if (iter_pair.first != iter_pair.second){
00550         // an atom exits in this grid
00551         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00552           if (out_iter != *(PointToHashIter[index])){
00553             d = getDistance((*out_iter).second,p); // distance
00554             if (d< distance) ReturnList.AddToFront((*out_iter).second); // index
00555           }
00556         }
00557       }
00558     } // end of neighbor1 cycles
00559   } else if (distance <= N2Dist){
00560     // Go through 27 one-neighbor grids and add up contributions
00561     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00562     hashmap_type::const_iterator out_iter;
00563     GridNumber TempGridNumber = createHash(p);
00564     for (int i = 0; i < 125; i++){
00565       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00566       if (iter_pair.first != iter_pair.second){
00567         // an atom exits in this grid
00568         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00569           if (out_iter != *(PointToHashIter[index])){
00570             d = getDistance((*out_iter).second,p); // distance
00571             if (d< distance) ReturnList.AddToFront((*out_iter).second); // index
00572           }
00573         }
00574       }
00575     } // end of neighbor2 cycles
00576   } else { // outside near neighbors, return all
00577     for (int i = 0; i < numPoints; i++){
00578       if (i != index){
00579         d = getDistance(i,p); // distance
00580         if (d < distance) ReturnList.AddToFront(i); // index
00581       }
00582     }
00583   }
00584
```

4.29.2.32 DLList< jVector > Grid::neighbor_p (const jVector & *p*, double *distance*) const

Definition at line 394 of file grid.C.

```
00394                                                                  {
00395   // Returns a list of **positions** to other points within a given
00396   // distance NB this calclulates and discards distances. (SLOW)
00397   DLList<jVector> ReturnList;
```

```
00398   double d;
00399   if (distance <= N1Dist) {
00400     // Go through 27 one-neighbor grids and add up contributions
00401     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00402     hashmap_type::const_iterator out_iter;
00403     GridNumber TempGridNumber = createHash(p);
00404     for (int i = 0; i < 27; i++){
00405       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00406       if (iter_pair.first != iter_pair.second){
00407         // an atom exits in this grid
00408         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00409           d = getDistance((*out_iter).second,p); // distance
00410           if (d< distance) ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00411         }
00412       }
00413     } // end of neighbor1 cycles
00414   } else if (distance <= N2Dist){
00415     // Go through 27 one-neighbor grids and add up contributions
00416     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00417     hashmap_type::const_iterator out_iter;
00418     GridNumber TempGridNumber = createHash(p);
00419     for (int i = 0; i < 125; i++){
00420       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00421       if (iter_pair.first != iter_pair.second){
00422         // an atom exits in this grid
00423         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00424           d = getDistance((*out_iter).second,p); // distance
00425           if (d< distance) ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00426         }
00427       }
00428     } // end of neighbor2 cycles
00429   } else { // outside near neighbors, return all
00430     for (int i = 0; i < numPoints; i++){
00431       d = getDistance(i,p); // distance
00432       if (d < distance) ReturnList.AddToFront(myPoints[i]); // jVector position
00433     }
00434   }
00435
```

4.29.2.33 DLList< jVector > Grid::neighbor_p (int *index*, double *distance*) const

Definition at line 342 of file grid.C.

```
00342                                                                {
00343   jVector p = myPoints[index];
00344     // Returns a list of **positions** to other points within a given
00345   // distance NB this calclulates and discards distances. (SLOW)
00346   DLList<jVector> ReturnList;
00347   double d;
00348   if (distance <= N1Dist) {
00349     // Go through 27 one-neighbor grids and add up contributions
00350     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00351     hashmap_type::const_iterator out_iter;
00352     GridNumber TempGridNumber = createHash(p);
00353     for (int i = 0; i < 27; i++){
00354       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor1[i]);
00355       if (iter_pair.first != iter_pair.second){
00356         // an atom exits in this grid
00357         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00358           if (out_iter != *(PointToHashIter[index])){
00359             d = getDistance((*out_iter).second,p); // distance
00360             if (d< distance) ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00361           }
```

```
00362         }
00363       }
00364     } // end of neighbor1 cycles
00365   } else if (distance <= N2Dist){
00366     // Go through 27 one-neighbor grids and add up contributions
00367     pair<hashmap_type::const_iterator, hashmap_type::const_iterator> iter_pair;
00368     hashmap_type::const_iterator out_iter;
00369     GridNumber TempGridNumber = createHash(p);
00370     for (int i = 0; i < 125; i++){
00371       iter_pair = GridHashMap.equal_range(TempGridNumber-Neighbor2[i]);
00372       if (iter_pair.first != iter_pair.second){
00373         // an atom exits in this grid
00374         for (out_iter = iter_pair.first; out_iter != iter_pair.second; ++out_iter){
00375           if (out_iter != *(PointToHashIter[index])){
00376             d = getDistance((*out_iter).second,p); // distance
00377             if (d< distance) ReturnList.AddToFront(myPoints[(*out_iter).second]); // jVector position
00378           }
00379         }
00380       }
00381     } // end of neighbor2 cycles
00382   } else { // outside near neighbors, return all
00383     for (int i = 0; i < numPoints; i++){
00384       if (i != index){
00385         d = getDistance(i,p); // distance
00386         if (d < distance) ReturnList.AddToFront(myPoints[i]); // jVector position
00387       }
00388     }
00389   }
00390
```

4.29.2.34 void Grid::setPosition (int *index_num*, const jVector & *NewPosition*)

Definition at line 208 of file grid.C.

Referenced by Molecule::Copy(), Molecule::DynRead(), Molecule::Input(), Molecule::InputBinary(), Molecule::Molecule(), Molecule::readXmol(), Molecule::setPosition(), and Molecule::setRandom().

```
00208                                                                  {
00209   // Moves the point with index == index_num to the new position
00210   // spoecified by NewPosition  This has to maintain the
00211
00212   myPoints[index_num] = NewPosition;
00213
00214
00215   // DEBUG
00216   // bounds check
00217   if ( (fabs(NewPosition[0]) > Length) ||
00218        (fabs(NewPosition[1]) > Length) ||
00219        (fabs(NewPosition[2]) > Length) ){
00220     cerr << "Grid::setPostion moved out of bounds: MaxGridLength = "
00221          << Length
00222          << " New Position = "
00223          << NewPosition
00224          << endl;
00225     raise (SIGQUIT);
00226   }
00227
00228   // Should check bounds here.  Not done now.  May cause errors if
00229   // max(fabs(NewPosition[0,1,2])) > Length.
00230   // need to call regrid if bounds are not OK.
00231
00232   GridNumber TempGridNumber = createHash(NewPosition);
```

```
00233   if (TempGridNumber !=  PointToGridNumArray[index_num]){
00234     // only unpdate hash if point moves to new grid.
00235     // delete old point in hash
00236     GridHashMap.erase(*(PointToHashIter[index_num]));
00237
00238     // add point in new grid to hash
00239     *PointToHashIter[index_num] = GridHashMap.insert(hashmap_type::value_type(TempGridNumber, index_num));
00240
00241     PointToGridNumArray[index_num] = TempGridNumber;
00242   }
00243
```

4.29.2.35 void Grid::setupNeighborLists (void) [private]

Definition at line 125 of file grid.C.
Referenced by Initialize().

```
00126 {
00127   register long x,y,z;
00128   int index = 0;
00129   for (z = -1; z < 2; z++){
00130     for (y = -1; y < 2; y++){
00131       for (x = -1; x < 2; x++){
00132         Neighbor1[index++] = z*GridDimensionSq + y*GridDimension + x;
00133       }
00134     }
00135   }
00136
00137   index = 0;
00138   for (z = -2; z < 3; z++){
00139     for (y = -2; y < 3; y++){
00140       for (x = -2; x < 3; x++){
00141         Neighbor2[index++] = z*GridDimensionSq + y*GridDimension + x;
00142       }
00143     }
00144   }
00145
```

4.29.3 Member Data Documentation

4.29.3.1 GridNumber Grid::GridDimension [private]

Definition at line 191 of file grid.h.

4.29.3.2 GridNumber Grid::GridDimensionSq [private]

Definition at line 192 of file grid.h.

4.29.3.3 hashmap_type Grid::GridHashMap [private]

Definition at line 180 of file grid.h.

4.29.3.4 double Grid::Length [private]

Definition at line 170 of file grid.h.

4.29.3.5 double Grid::N1Dist [private]

Definition at line 172 of file grid.h.

4.29.3.6 double Grid::N2Dist [private]

Definition at line 173 of file grid.h.

4.29.3.7 GridNumber Grid::Neighbor1[27] [private]

Definition at line 213 of file grid.h.

4.29.3.8 GridNumber Grid::Neighbor2[125] [private]

Definition at line 214 of file grid.h.

4.29.3.9 DLList< GridNumber > Grid::NeighborGridList [private]

Definition at line 185 of file grid.h.

4.29.3.10 DLList< int > Grid::PointIndexList [private]

Definition at line 186 of file grid.h.

4.29.3.11 GridNumber * Grid::PointToGridNumArray [private]

Definition at line 188 of file grid.h.

4.29.3.12 hashmap_type::iterator ** Grid::PointToHashIter [private]

Definition at line 196 of file grid.h.

4.29.3.13 double Grid::TwiceLength [private]

Definition at line 171 of file grid.h.

4.29.3.14 double Grid::delta [private]

Definition at line 174 of file grid.h.

4.29.3.15 pointList Grid::myPoints [private]

Definition at line 177 of file grid.h.

4.29.3.16 int Grid::numPoints [private]

Definition at line 175 of file grid.h.

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

- **grid.h**
- **grid.C**

4.30 gwNumerics Class Reference

`#include <gw_numerics.h>`

Public Methods

- **gwNumerics** ()
- ~**gwNumerics** ()

Static Public Methods

- void **SolveVector** (const **jVector** &gradient, const **jVector** &\hessian, **jVector** &result)
- double **CalculateDerivative** (const **jVector** &direction, const**jVector** &gradient)
- bool **CubicLineSearch** (const **jVector** &position, const **jVector** &\direction, const double &derivative, const**ConstraintManager** &CM, int atom_index,**jVector** &NewPosition, double &qmin)
- **jVector SetHCPosition2** (const **jVector** &start_position, const **jVector** &curr_position,\const **jVector** &direction, **ConstraintManager** &\CM, int atom_index, double qmin_dist, double &max_lpert_r)
- void **addVDWSegments** (**Segments** &Seglist, const **jVector** &curr_position,\const **jVector** &direction, **ConstraintManager** &\CM, int atom_index, double offset=0)
- void **addLPERTSegments** (**Segments** &Seglist, const **jVector** &curr_position,\const **jVector** &direction, **ConstraintManager** &\CM, int atom_index, double qmin_dist,\const **jVector** &start_position, const **jVector** &new_position, double &max_r)
- double **LinearThreshold** (const double &stress, double *RadArray)
- **Segment LineIntersectSphere** (const **jVector** &atom_position, const double &\radius, const **jVector** &vec_position, const**jVector** &atom_direction)
- void **bfgs_step** (const **jVector** &x, const **jVector** &xp, const **jVector** &gf, const **jVector** &gfp, **jVector** &H, **jVector** &direction)
- void **LocalPert** (int I, **ConstraintManager** &CM, double min_dist, **jVector** start_position)
- int **SpreadR** (int I, double radius, **DLList**< int > Moved, int depth, **ConstraintManager** &CM, **jVector** start_position, const **jVector** pref_dir)
- **DLList**<int> **CheckViol** (int I, **DLList**< int > NeighborList, bool UsePrevious, const **ConstraintManager** &CM)
- int **MoveOnSingleViol** (int I, double radius, int otherAtom, **ConstraintManager** &CM)
- **jVector CalcForce** (int I, double radius, **DLList**< int > ViolList, **DLList**< int > NeighborList, **DLList**< int > Moved, const **ConstraintManager** &CM, const **jVector** &start_position, double min_dist, const **jVector** pref_dir)
- void **reset_LP_Count** (void)

Static Public Attributes

- int **NumLP** = 0
- int **NumNotLP** = 0
- bool **UseOnlyMoved** = true

  Flag to set vdw exclusion based on previously moved atoms only.

- double **ls_dist_tolerance** = 0.0001
- double **ls_succ_tolerance** = 0.001

- double **ls_deriv_tolerance** = 0.0000001
- bool **LocalPerturb** = true

  Setting for local perturbation activation.

- int **MaxNCuts** = 10

  Maximum number of cuts used in the cubic line search.

- double **LPertFactor**

  Multiplication factor for local pert radius.

- double **MaxAbsDistRes** = -1 * INF

  the max residual over distance constraints.

- double **AveAbsDistRes** = 0

  the average residual over distance constraints.

- double **MinDistVariance** = INF

  the minimum variance fron distance constraint data.

4.30.1 Detailed Description

//

File: **gw_numerics.h**

Created by: Jon Dugan

This class is never instantiated. It containts static, public functions available for use by other classes (in this case GRAPE Jobs). Several of the numerical techniques originally came from Glenn Williams, hence the 'gw_' in the name.

//

Definition at line 33 of file gw_numerics.h.

4.30.2 Constructor & Destructor Documentation

4.30.2.1 gwNumerics::gwNumerics ()

Definition at line 3 of file gw_numerics.C.

```
00004 {
00005
```

4.30.2.2 gwNumerics::~gwNumerics ()

Definition at line 7 of file gw_numerics.C.

```
00008 {
00009
```

4.30.3 Member Function Documentation

4.30.3.1 jVector gwNumerics::CalcForce (int *I*, double *radius*, DLList< int > *ViolList*, DLList< int > *NeighborList*, DLList< int > *Moved*, const ConstraintManager & *CM*, const jVector & *start_position*, double *min_dist*, const jVector *pref_dir*) [static]

4.30.3.2 double gwNumerics::CalculateDerivative (const jVector & *direction*, const\jVector & *gradient*) [static]

Returns the derivative of the evaluation function at a position. GGG This return variable is used to check for convergence in the iterations GGG

4.30.3.3 DLList<int> gwNumerics::CheckViol (int *I*, DLList< int > *NeighborList*, bool *UsePrevious*, const ConstraintManager & *CM*) [static]

4.30.3.4 bool gwNumerics::CubicLineSearch (const jVector & *position*, const jVector &\ *direction*, const double & *derivative*, const\ConstraintManager & *CM*, int *atom_index*, \jVector & *NewPosition*, double & *qmin*) [static]

Returns bool flag indication success(true) or failure (false). The function also resets the value in 'NewPosition' to the best position on the line if the search was successful.

Referenced by gwRelaxAtom::run().

4.30.3.5 Segment gwNumerics::LineIntersectSphere (const jVector & *atom_position*, const double &\ *radius*, const jVector & *vec_position*, const\jVector & *atom_direction*) [static]

using the vec_position and vec_direction, a line in three space is tested if an atom at atom_position with radius specified by radius intersects that line. **Segment** (p. 480) returned has units of direction.mag()

Referenced by addLPERTSegments().

4.30.3.6 double gwNumerics::LinearThreshold (const double & *X*, double * *Array*) [static]

Determines the 'radius' of an atom for hard constraint purposes using the linear interpolation method between the max and min settings in RadArray.

Definition at line 466 of file gw_numerics.C.

```
00467 {
00468   // The Array specifies the linear function that maps one space
00469   // into another, like from X to Y.  Array[0] = Xmin,
00470   // Array[1] = Yval for X < Xmin.  Array[2]= Xmax-Xmin, and Array[3]=
00471   // Yval for X > Xmax.  Linear interpolation is done if X is between
00472   // Xmin and Xmax.
00473
00474   if (X <= Array[0]) return Array[1];
00475   if (X >= Array[0]+Array[2]) return Array[3]; // Array[2]==0 => step fntn
00476   if (Array[2]!=0){ //just in case
00477     return Array[1]+(Array[3]-Array[1])*((X-Array[0])/Array[2]);
00478   }
00479
```

4.30.3.7 void gwNumerics::LocalPert (int *I*, ConstraintManager & *CM*, double *min_dist*, jVector *start_position*) [static]

4.30.3.8 int gwNumerics::MoveOnSingleViol (int *I*, double *radius*, int *otherAtom*, ConstraintManager & *CM*) [static]

4.30.3.9 jVector gwNumerics::SetHCPosition2 (const jVector & *start_position*, const jVector & *curr_position*, \const jVector & *direction*, ConstraintManager &\ *CM*, int *atom_index*, double *qmin_dist*, double & *max_lpert_r*) [static]

This returns a [3x1] vector used to move an atom in its final iteration, conforming to the Van Der Waal constraints.

Definition at line 88 of file gw_numerics.C.

Referenced by gwRelaxAtom::run().

```
00090                                                                {
00091
00092
00094   //    overview
00095   //
00096   // add vdw to seglist
00097   // add lpert to seglist
00098   // copy seglist
00099   // check viol
00100   // if viol -- loop:
00101   //   // get neighbors at best
00102   //   // add vdw
00103   //   // check viol
00104   // until !viol
00105   //
00107
00108   Segments SegList;
00109   Segment TempSegment;
00110   double func_eval1, func_eval2;
00111
00112   jVector new_position = curr_position +direction*qmin_dist;
00113
00114 //   cerr << "*1 new_position =>" << new_position << endl;
00115 //   cerr << "*1 curr_position =>" << curr_position << endl;
00116
00117 //   cerr << "new_position" << new_position << endl;
00118 //   cerr << "curr_position" << curr_position << endl;
00119 //   cerr << "qmin_dist"  << qmin_dist << endl;
00120
00121 #ifdef GNOMAD_OUTPUT
00122       printf("  Initial qmin:\n");
00123       printf("    initial qmin = %lf\n", qmin_dist);
00124 #endif
00125
00126
00127   // add vdw to seglist
00128       addVDWSegments(SegList, curr_position,  \
00129                         direction, CM, atom_index, 0);
00130
00131         //  cerr << "*2 new_position =>" << new_position << endl;
00132
00133   // add lpert to seglist
00134   if (LocalPerturb)
00135     addLPERTSegments (SegList, curr_position,  \
00136                       direction, CM, atom_index, qmin_dist, \
```

```
00137                       start_position, new_position, max_lpert_r);
00138
00139   //  cerr << "*3 new_position =>" << new_position << endl;
00140
00141   // check viol  -- write seglist.qsortviol
00142   TempSegment = SegList.QSortReduceViol(qmin_dist);
00143
00144   if ((TempSegment.start == 0) && (TempSegment.end == 0)){
00145     // no violation
00146
00147 #ifdef GNOMAD_OUTPUT
00148     printf("  After vdw_segs\n");
00149     //    printf("    lbound, ubound = %lf %lf\n", *lbound, *ubound);
00150     //    printf("    num_segments = %d\n", *num_segments);
00151     printf ("Segment List:\n");
00152     for(int ixx=0; ixx < SegList.GetNum(); ixx++)
00153       printf("      start_ineq, end_ineq = %lf %lf\n", SegList.GetSeg(ixx).start, SegList.GetSeg(ixx).end );
00154     printf("\n");
00155
00156     printf("  After viol_check\n");
00157     printf("    qmin = %lf\n\n", qmin_dist);
00158 #endif
00159
00160     //    cerr << "*4 new_position =>" << new_position << endl;
00161     return new_position;
00162
00163   } else {     // BEGIN Violation else
00164
00165
00166     while ((TempSegment.start != 0) && (TempSegment.end != 0)){
00167       // BEGIN location loop:
00168
00169       //find current best
00170       func_eval1 = CM.stress(atom_index, curr_position+direction*TempSegment.start);
00171       func_eval2 = CM.stress(atom_index, curr_position+direction*TempSegment.end);
00172       qmin_dist = (func_eval1 < func_eval2)? TempSegment.start : TempSegment.end;
00173
00174       //change qmin_dist to be there
00175       new_position = curr_position + (direction*qmin_dist);
00176
00177       //      cerr << "*5 new_position =>" << new_position << endl;
00178 //       cerr << "*5 direction =>" << direction << endl;
00179 //       cerr << "*5 direction.mag() =>" << direction.mag() << endl;
00180 //       cerr << "*5 qmin =>" << qmin_dist << endl;
00181
00182       // add new vdw
00183       addVDWSegments(SegList, new_position,  \
00184                      direction, CM, atom_index, qmin_dist);
00185
00186       //      cerr << "*6 new_position =>" << new_position << endl;
00187
00188       //  check viol
00189       TempSegment = SegList.QSortReduceViol(qmin_dist);
00190
00191       // if no violation at current qmin_dist, then TempSegment will be 0,0 -- break out of loop.
00192
00193     }   // END location loop
00194
00195
00196 #ifdef GNOMAD_OUTPUT
00197     printf("  After vdw_segs\n");
00198     //    printf("    lbound, ubound = %lf %lf\n", *lbound, *ubound);
00199     //    printf("    num_segments = %d\n", *num_segments);
00200     printf ("Segment List:\n");
00201     for(int ixx=0; ixx < SegList.GetNum(); ixx++)
```

```
      printf("      start_ineq, end_ineq = %lf %lf\n", SegList.GetSeg(ixx).start, SegList.GetSeg(ixx).end );
    printf("\n");

    printf("  After viol_check\n");
    printf("    qmin = %lf\n\n", qmin_dist);
#endif

    //    cerr << "*7 new_position =>" << new_position << endl;
    return new_position;

  }

```

4.30.3.10 void gwNumerics::SolveVector (const jVector & *gradient*, const jVector &\ *hessian*, jVector & *result*) [static]

Result pointer is a [3x1] array of doubles that represent are the solution to Hessian (x1,y1,z1) = Gradient. GGG in Gnomad, x1,y1,z1 (result∗) is variable p GGG

Referenced by bfgs_step().

4.30.3.11 int gwNumerics::SpreadR (int *I*, double *radius*, DLList< int > *Moved*, int *depth*, ConstraintManager & *CM*, jVector *start_position*, const jVector *pref_dir*) [static]

4.30.3.12 void gwNumerics::addLPERTSegments (Segments & *SegList*, const jVector & *curr_position*, \const jVector & *direction*, ConstraintManager &\ *CM*, int *atom_index*, double *qmin_dist*, \const jVector & *start_position*, const jVector & *new_position*, double & *max_r*) [static]

Definition at line 308 of file gw_numerics.C.

```
                                                                                    {


#ifdef GNOMAD_OUTPUT
  printf("  In pert_seg\n");
  printf("    initial x0, y0, z0 = %lf  %lf  %lf\n", start_position[0], start_position[1], start_position[2]);
  printf("    initial x, y, z = %lf  %lf  %lf\n", curr_position[0], curr_position[1], curr_position[2]);
  printf("    initial direction vector = %lf  %lf  %lf\n", direction[0], direction[1], direction[2]);
  //  printf("    initial lbound, ubound = %lf %lf\n", *lbound, *ubound);
  //  printf("    num_segments = %d\n\n", *num_segments);
#endif

#ifdef JOB_VERBOSE
  cerr << "addLPERTSegments: starting " << endl;
#endif

  Segment TempSegment;
  //  double radius = LinearThreshold(sqrt(CM.stress(atom_index, curr_position)), LPertRadius);
  //double radius = sqrt(0.5*CM.stress(atom_index, curr_position));
  double radius = sqrt(0.5*MinDistVariance*CM.stress(atom_index, new_position));

  if (radius > max_r){
#ifdef JOB_VERBOSE2
    cerr << "addLPERTSegments: pert radius =    " << radius << endl;
    cerr << "RESETTING PERT RADIUS ARBITRARILY to " << radius << endl ;
#endif
    radius = max_r;
```

```
00339   }
00340
00341 #ifdef JOB_VERBOSE2
00342   cerr << "addLPERTSegments: pert stress =    " << CM.stress(atom_index, new_position)  << endl;
00343   cerr << "addLPERTSegments: pert dif vector =" << curr_position-start_position << endl;
00344   cerr << "addLPERTSegments: pert radius =    " << radius << endl;
00345   cerr << "addLPERTSegments: pert max_radius cutoff =    " << max_r << endl;
00346   cerr << "addLPERTSegments: pert mindistvar =    " << MinDistVariance << endl;
00347 #endif
00348   //cerr << "r " << radius << endl;
00349   //radius = LinearThreshold(log10(CM.stress(atom_index, curr_position)+1), LPertRadius);
00350   //radius = LinearThreshold(LPertRadius[2]-(1/(CM.stress(atom_index, curr_position)+LPertRadius[0])), LPertRadius);
00351   // radius = radius * LPertFactor;
00352   if (radius>0) {
00353     TempSegment = LineIntersectSphere(start_position, radius, curr_position, direction);
00354
00355 #ifdef JOB_VERBOSE2
00356     cerr << "addLPERTSegments: pert segement =" << TempSegment << endl;
00357 #endif
00358
00359     if (TempSegment.start < -1e8 && TempSegment.end > 1e8) { // two number from Glenn's ass
00360       return;
00361     }
00362
00363     if ((qmin_dist > TempSegment.start) && (qmin_dist < TempSegment.end)) {
00364 #ifdef JOB_VERBOSE2
00365       cerr << "addLPERTSegments: segment caused perturbation" << endl;
00366 #endif
00367       NumLP++;
00368       SegList.Add(TempSegment);
00369       // here is where Local Pert has bumped atom number atom_index'
00370     } else {
00371 #ifdef JOB_VERBOSE2
00372       cerr << "addLPERTSegments: pert segment irrelevant" << endl;
00373 #endif
00374       NumNotLP++;
00375     }
00376
00377   }
00378
```

4.30.3.13 void gwNumerics::addVDWSegments (Segments & *SegList*, const jVector & *curr_position*, \const jVector & *direction*, ConstraintManager &\ *CM*, int *atom_index*, double *offset* = 0) [static]

Definition at line 217 of file gw_numerics.C.

```
00219                                    {
00220
00221 #ifdef JOB_VERBOSE
00222   cerr << "addVDWSegments: starting " << endl;
00223   cerr << "addVDWSegments: offset = "<<offset << endl;
00224   if (direction.nrm() == 0)
00225     cerr << "addVDWSegments: Zero dir! (abort)" << endl;
00226 #endif
00227
00228   if (direction.nrm() == 0)
00229     return ;
00230
00231
00232   DLList<int> NeighborList = CM.approx_neighbor_i(curr_position, 3.7);
00233   DLList<int>::ConstIterator Nei_Iter;
```

```
00234   int other_index;
00235   double msd;  // Minimum Separation Distance (center to center)
00236   Segment TempSegment;
00237
00238   for(Nei_Iter = NeighborList.GetConstIterator(); Nei_Iter; ++Nei_Iter){
00239     // BEGIN iterate over local neighbors
00240     other_index = *Nei_Iter;
00241
00242 #ifdef JOB_VERBOSE2
00243     cerr << "addVDWSegments: checking atom " << other_index << endl;
00244 #endif
00245
00246     //    msd = LinearThreshold((CM.stress(other_index)),VDWRadius);
00247     //msd = 3;
00248     //msd    moved into loop to prevent msd on atom = atom
00249
00250     //    cerr << "Bonds for atom " << atom_index << ": " << CM.getMolecule().GetBonds(atom_index) << endl;
00251
00252     // DEBUG
00253     //cerr << "vdw atom:     "<< other_index;
00254     //cerr << "vdw msd:   "<< msd;
00255     //cerr << "vdw position: "<< (CM.getMolecule())(other_index) << endl;
00256
00257     // here use the Moved() atom info to only check VDW against moved atoms
00258     if ((other_index != atom_index) && (CM.constraintON(other_index))) {
00259       if ((UseOnlyMoved == false)
00260           || ((UseOnlyMoved == true)
00261               && (CM.getAtom(other_index).Moved() == true)
00262               ))
00263         {
00264           msd = (CM.getMolecule()).MSD(atom_index, other_index);
00265
00266 #ifdef JOB_VERBOSE2
00267           cerr << "addVDWSegments: atom " << other_index <<
00268             " passed move and self-atom criteria. msd = " << msd << endl;
00269
00270           cerr << "x            " << curr_position << endl;
00271           cerr << "other atom x " << (CM.getMolecule())(other_index) << endl;
00272           cerr << "dir          " << direction << endl;
00273 #endif
00274
00275           TempSegment = LineIntersectSphere((CM.getMolecule())(other_index),
00276                                             msd, curr_position ,direction);
00277           // returns segment on 0,0 if not intersect
00278           //cerr << "vdw segment: " << TempSegment << endl;
00279
00280 #ifdef JOB_VERBOSE2
00281           cerr << "addVDWSegments: segment returned " << TempSegment
00282                << " for atom " << other_index << "; d = "
00283                << euclid ((CM.getMolecule())(other_index), curr_position)
00284                << endl;
00285 #endif
00286
00287           if (offset){
00288             TempSegment.start += offset;
00289             TempSegment.end += offset;
00290 #ifdef JOB_VERBOSE2
00291           cerr << "addVDWSegments: segment offset to " << TempSegment
00292                << " for atom " << other_index << endl;
00293 #endif
00294
00295           }
00296           if (TempSegment.start != TempSegment.end) SegList.Add(TempSegment);
00297         }
00298     }
```

```
00299   }
00300
```

4.30.3.14 void gwNumerics::bfgs_step (const jVector & *x*, const jVector & *xp*, const jVector & *gf*, const jVector & *gfp*, jVector & *H*, jVector & *direction*) [static]

Definition at line 921 of file gw_numerics.C.

Referenced by gwRelaxAtom::run().

```
00923                                                    {
00924
00925 //      cerr << "D: -  H-row0= " << H[0] << " " << H[1] << " " << H[2] << endl;
00926 //      cerr << "D: -  H-row1= " << H[1] << " " << H[3] << " " << H[4] << endl;
00927 //      cerr << "D: -  H-row2= " << H[2] << " " << H[4] << " " << H[5] << endl;
00928 //      cerr << "D:  -  x= " << x << endl;
00929 //      cerr << "D:  -  xp= " << xp << endl;
00930 //      cerr << "D:  -  gf= " << gf << endl;
00931 //      cerr << "D:  -  gfp= " << gfp << endl;
00932 //      cerr << "D:  -  p= " << direction << endl;
00933   // solves for p, the direction vector
00934
00935   register int i;
00936   // double Acopy[9], bcopy[3], d1, d2, hs[3], h1[6], h2[6], s[3], st, temp, v[3];
00937   double st, d1, d2, temp, h1[6], h2[6], hs[3];
00938   jVector Acopy(9,0);
00939   jVector s;
00940   jVector v;
00941
00942   st = 0.0;
00943
00944   s = x - xp;
00945 //   cerr << "D:  -  s= " << s << endl;
00946   v = gf - gfp;
00947 //   cerr << "D:  -  v= " << v << endl;
00948
00949   d1 = 1/(v*s);
00950 //   cerr << "D:  -  d1= " << d1 << endl;
00951
00952   h1[0] = d1*v[0]*v[0];
00953   h1[1] = d1*v[0]*v[1];
00954   h1[2] = d1*v[0]*v[2];
00955   h1[3] = d1*v[1]*v[1];
00956   h1[4] = d1*v[1]*v[2];
00957   h1[5] = d1*v[2]*v[2];
00958
00959   temp = H[1]*s[0]*s[1] + H[2]*s[0]*s[2] + H[4]*s[1]*s[2];
00960   temp *= 2.0;
00961   d2 = H[0]*s[0]*s[0] + H[3]*s[1]*s[1] + H[5]*s[2]*s[2];
00962   d2 += temp;
00963   d2 = 1.0/d2;
00964
00965 //   cerr << "D:  -  d2= " << d2 << endl;
00966
00967   hs[0] = H[0]*s[0] + H[1]*s[1] + H[2]*s[2];
00968   hs[1] = H[1]*s[0] + H[3]*s[1] + H[4]*s[2];
00969   hs[2] = H[2]*s[0] + H[4]*s[1] + H[5]*s[2];
00970
00971   h2[0] = d2*hs[0]*hs[0];
00972   h2[1] = d2*hs[0]*hs[1];
00973   h2[2] = d2*hs[0]*hs[2];
```

```
00974   h2[3] = d2*hs[1]*hs[1];
00975   h2[4] = d2*hs[1]*hs[2];
00976   h2[5] = d2*hs[2]*hs[2];
00977
00978   for(i=0; i<6; i++) H[i] = H[i] + h1[i] - h2[i];
00979
00980   Acopy[0] = H[0];
00981   Acopy[1] = H[1];
00982   Acopy[2] = H[2];
00983   Acopy[3] = H[1];
00984   Acopy[4] = H[3];
00985   Acopy[5] = H[4];
00986   Acopy[6] = H[2];
00987   Acopy[7] = H[4];
00988   Acopy[8] = H[5];
00989
00990
00991   gwNumerics::SolveVector((-1)*gf, Acopy, direction);
00992
00993
```

4.30.3.15 void gwNumerics::reset_LP_Count (void) [inline, static]

Definition at line 191 of file gw_numerics.h.

```
00191                                          {
00192     NumLP = NumNotLP = 0;
00193
```

4.30.4 Member Data Documentation

4.30.4.1 double gwNumerics::AveAbsDistRes = 0 [static]

the average residual over distance constraints.

Definition at line 11 of file globals.C.

4.30.4.2 double gwNumerics::LPertFactor [static]

Multiplication factor for local pert radius.

Definition at line 223 of file gw_numerics.h.

4.30.4.3 bool gwNumerics::LocalPerturb = true [static]

Setting for local perturbation activation.

Definition at line 20 of file globals.C.

4.30.4.4 double gwNumerics::MaxAbsDistRes = -1 * INF [static]

the max residual over distance constraints.

Definition at line 10 of file globals.C.

4.30.4.5 int gwNumerics::MaxNCuts = 10 [static]

Maximum number of cuts used in the cubic line search.

Definition at line 21 of file globals.C.

4.30.4.6 double gwNumerics::MinDistVariance = INF [static]

the minimum variance fron distance constraint data.

Definition at line 12 of file globals.C.

4.30.4.7 int gwNumerics::NumLP = 0 [static]

Definition at line 22 of file globals.C.

4.30.4.8 int gwNumerics::NumNotLP = 0 [static]

Definition at line 23 of file globals.C.

4.30.4.9 bool gwNumerics::UseOnlyMoved = true [static]

Flag to set vdw exclusion based on previously moved atoms only.

Definition at line 19 of file globals.C.

4.30.4.10 double gwNumerics::ls_deriv_tolerance = 0.0000001 [static]

Tolerance of convergence. This value is used to test the derivative of the evaluation function duringthe line search. (tau)

Definition at line 8 of file globals.C.

4.30.4.11 double gwNumerics::ls_dist_tolerance = 0.0001 [static]

Epsilon of useless movement. If an atom is moved less than this amount, it is assumed to be a trivial movement. (1e-4)

Definition at line 7 of file globals.C.

4.30.4.12 double gwNumerics::ls_succ_tolerance = 0.001 [static]

Tolerance value used to test if the line search function was successful. (alph in glenn wiliams code)

Definition at line 6 of file globals.C.

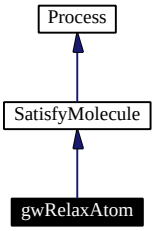
The documentation for this class was generated from the following files:

- **gw_numerics.h**
- **globals.C**
- **gw_numerics.C**

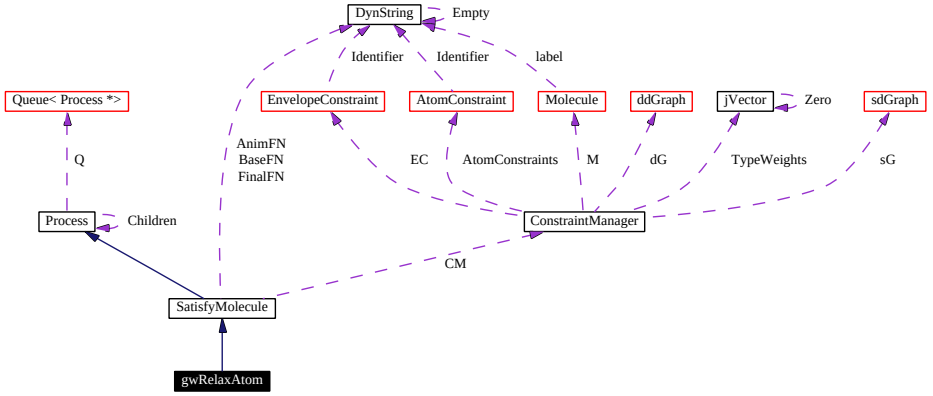
4.31 gwRelaxAtom Class Reference

`#include <gw_relax_atom.h>`

Inheritance diagram for gwRelaxAtom



Collaboration diagram for gwRelaxAtom:



Public Methods

- **gwRelaxAtom** (const gwRelaxAtom &ra)

  Copy Constructor.

- **gwRelaxAtom** (**ConstraintManager** &cm, int index)

  Constructor takes a reference to a constraint manager, and the index of the atom.

- **~gwRelaxAtom** ()
- int **run** ()
- int **getIndex** (void) const

  return index of the atom the class is acting on.

Static Public Attributes

- int **max_iterations** = 40

  Maximum nuber of times to perform linesearch while updating atom.

Private Attributes

- int **atom_index**

  The index of the atom in the molecule this process is associated with.

4.31.1 Detailed Description

//

File: **gw_relax_atom.h**

This class re-implements exactly the atom relaxtion code from Glenn Williams' Gnomad software. A modified Gauss Newton using the gradient and Hessian determine the optimal direction of travel for the atom, and a 2nd oder polynomial line search determines the best new position. Hard constraint provides both Van del Waals excusion and the ability to perform local pertubation.

//

Definition at line 25 of file gw_relax_atom.h.

4.31.2 Constructor & Destructor Documentation

4.31.2.1 gwRelaxAtom::gwRelaxAtom (const gwRelaxAtom & *ra*)

Copy Constructor.

Definition at line 4 of file gw_relax_atom.C.

```
00005   : SatisfyMolecule(*(ra.CM))
00006 {
00007   atom_index = ra.atom_index;
00008
```

4.31.2.2 gwRelaxAtom::gwRelaxAtom (ConstraintManager & *cm*, int *temp_index*)

Constructor takes a reference to a constraint manager, and the index of the atom.

Definition at line 10 of file gw_relax_atom.C.

```
00011   : SatisfyMolecule(cm)
00012 {
00013   atom_index = temp_index;
00014   // gwRA assumes that static double ls_deriv_tolerance,
00015   // static double ls_dist_tolerance,
00016
```

4.31.2.3 gwRelaxAtom::~gwRelaxAtom ()

Definition at line 18 of file gw_relax_atom.C.

```
00019 {
00020   // Do *not* delete constraint manager (or M for that matter)!
00021
```

4.31.3 Member Function Documentation

4.31.3.1 int gwRelaxAtom::getIndex (void) const [inline]

return index of the atom the class is acting on.
Definition at line 71 of file gw_relax_atom.h.

```
00071                                          {return atom_index;}
00072
```

4.31.3.2 int gwRelaxAtom::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).
Definition at line 23 of file gw_relax_atom.C.

```
00024 {
00025
00026 #ifdef JOB_VERBOSE
00027   cerr <<endl<< "Start: \tgwRelaxAtom::run() atom=" << atom_index << endl;
00028 #endif
00029
00030 #ifdef JOB_VERBOSE2
00031   cerr << "Value: \tgwRelexAtom::run() priority atom " << atom_index << "before ="<< getPriority() << endl;
00032 #endif
00033
00034
00035 // #ifdef GNOMAD_OUTPUT
00036 //   printf("  In move_atom for atom %d\n\n", atom_index);
00037 // #endif
00038
00039   //  cerr << ";" << atom_index ;
00040
00041   jVector hessian(6,0.0);
00042   // 3x3 Hessian symmetric matrix of partial derivatives dr/di*dr/dj, where
00043   // i and j iterate over x,y,z dimension, and r is the residual
00044   // of this atom.
00045   // 0 1 2
00046   // 1 3 4
00047   // 2 4 5
00048
00049   jVector gradient(3,0.0);
00050   // Gradient vector.
00051
00052   jVector prev_gradient(3,0.0);
00053   // previous Gradient vector.
00054
00055   jVector direction(3,0.0);
00056   // Direction atom moves during each step inside iteration
00057
00058   jVector prev_direction(3,0.0);
00059   // previous Direction atom moves during each step inside iteration
00060
00061   jVector start_position = CM->getPosition(atom_index);
00062
00063   jVector step_position = start_position;
00064   // pulls the x,y,z coordinate of the atom before relaxation.
00065
00066   jVector prev_position = start_position;
```

```
00067
00068   jVector new_position = start_position;
00069
00070    //  jVector temp_vector(12,0.0);
00071   // FIX  a hack to hold the dStress(int) returnval.
00072
00073   double step_length = 0;
00074   double new_step_length = 0;
00075   double derivative;
00076   int iteration = 0;
00077   register int i;
00078   bool ContinueFlag = true;
00079
00080   for(i=0; i<3; i++) gradient[i] = 0.0;  // reset jVectors
00081   for(i=0; i<6; i++) hessian[i] = 0.0;
00082
00083 #ifdef GNOMAD_OUTPUT
00084   printf("    x0, y0, z0 = %lf, %lf, %lf\n", start_position[0], start_position[1], start_position[2]);
00085   printf("    f0 = %lf\n\n", CM->stress(atom_index));
00086 #endif
00087
00088   // 'iteration' loop
00089   while (ContinueFlag){
00090
00091 #ifdef JOB_VERBOSE2
00092     cerr << "Working\tgwRelaxAtom::run() start iteration number " << iteration << endl << endl;
00093     cerr << "Value: \tgwRelaxAtom::run() step_position (start of iteration)=  " << step_position <<endl;
00094     cerr << "Value: \tgwRelaxAtom::run() prev_position (start of iteration)=  " << prev_position <<endl;
00095     cerr << "Stress = "<< setprecision(25) <<CM->stress(atom_index,step_position) << setprecision(8)  <<endl;
00096 #endif
00097
00098
00099     if (iteration >= max_iterations) {
00100       //      cerr << "X";
00101       break;  // break condition on iteration count
00102     }
00103
00104     gradient = CM->dStress(atom_index, step_position);
00105
00106     if (iteration == 0) {
00107       hessian[0] = hessian[3] =  hessian[5] =  1;
00108       direction = (-1)*gradient;
00109
00110     } // first iteration only
00111     else { // all other iters
00112
00113       gwNumerics::bfgs_step(step_position, prev_position,
00114                             gradient, prev_gradient,
00115                             hessian, direction);
00116       // sets hessian and direction
00117     }
00118
00119     derivative = direction*gradient;
00120 #ifdef JOB_VERBOSE2
00121     cerr << "Value: \tgwRelaxAtom::run() gradient = " << gradient << endl;
00122     cerr << "Value: \tgwRelaxAtom::run() hessian0 = "
00123          << hessian[0] << "\t" << hessian[1]  << "\t" << hessian[2]  << "\t" << endl;
00124     cerr << "Value: \tgwRelaxAtom::run() hessian1 = "
00125          << hessian[1]  << "\t" << hessian[3]  << "\t" << hessian[4] << endl;
00126     cerr << "Value: \tgwRelaxAtom::run() hessian2 = "
00127          << hessian[2]  << "\t" << hessian[4]  << "\t" << hessian[5] << endl;
00128     cerr << "Value: \tgwRelaxAtom::run() direction =  " << direction << endl;
00129     cerr << "Value: \tgwRelaxAtom::run() derivative =  " << derivative << endl;
00130 #endif
00131
```

```
00132     if (fabs(derivative) < gwNumerics::ls_deriv_tolerance) {
00133 #ifdef JOB_VERBOSE2
00134       if (iteration == 0) {prev_direction = direction;}
00135       cerr << "break on derivative too small" << endl;
00136     cerr << "Value: \tgwRelaxAtom::breaking step_position =  " << step_position <<endl;
00137     cerr << "Value: \tgwRelaxAtom::breaking prev_position =  " << prev_position <<endl;
00138 #endif
00139       break; // break condition on derivative
00140     }
00141
00142     ContinueFlag = gwNumerics::CubicLineSearch(step_position, direction, \
00143                                   derivative, *CM, \
00144                                   atom_index, new_position, new_step_length);
00145     // both new_position and step_length are reset here
00146
00147 #ifdef JOB_VERBOSE2
00148     cerr << "Line search done, ContinueFlag = "<< ContinueFlag << endl;
00149     cerr << "Value: \tgwRelaxAtom::run() step_length =  " << step_length << endl;
00150     cerr << "Value: \tgwRelaxAtom::run() new_step_length =  " << new_step_length << endl;
00151     cerr << "Value: \tgwRelaxAtom::run() new_position =  " << new_position <<endl;
00152 #endif
00153
00154     if (ContinueFlag) {  // successful line search
00155       prev_position = step_position;
00156       step_position = new_position;
00157       prev_direction = direction;
00158       prev_gradient = gradient;
00159       step_length = new_step_length;
00160 //       cerr << "Value: \tgwRelaxAtom::breaking step_length =  " << step_length <<endl;
00161 //       cerr << "Value: \tgwRelaxAtom::breaking new_step_length =  " << new_step_length <<endl;
00162
00163       //     } else {
00164 //     if (iteration == 0){
00165 //      prev_direction = direction;
00166 //      WARN ("debug from gnomad");
00167 //      cerr << "prev_direction =  " << prev_direction << endl;
00168 //      cerr << "direction =       " << direction << endl;
00169 //     }
00170     }
00171
00172 #ifdef JOB_VERBOSE2
00173     cerr << "Value: \tgwRelaxAtom::run() step_position (end of iteration)=  " << step_position << endl;
00174     cerr << "Working\tgwRelaxAtom::run() done iteration number " << iteration << endl << endl;
00175 #endif
00176
00177     iteration++;
00178   } // end of inner iteration loop
00179
00180 // #ifdef GNOMAD_OUTPUT
00181 //     printf("  In vdw_segs\n");
00182 //     printf("    initial x, y, z = %lf  %lf  %lf\n", prev_position[0], prev_position[1], prev_position[2]);
00183 //     printf("    initial direction vector = %lf  %lf  %lf\n", prev_direction[0], prev_direction[1], prev_direction[2
00184 //   //  printf("    initial lbound, ubound = %lf %lf\n", *lbound, *ubound);
00185 //   //  printf("    num_segments = %d\n\n", *num_segments);
00186 // #endif
00187
00188
00189   //  if (step_length != 1) cerr  << step_length << endl;
00190
00191   /* do hard constraint (VDW) check. */
00192   //jVector ddd = prev_position;
00193
00194
00195
00196 //     cerr << "Value: \tgwRelaxAtom::after iterator step_position =  " << step_position <<endl;
```

```
00197 //     cerr << "Value: \tgwRelaxAtom::after iterator prev_position =  " << prev_position <<endl;
00198 //     cerr << "Value: \tgwRelaxAtom::after iterator step_length =  " << step_length <<endl;
00199
00200     new_position = gwNumerics::SetHCPosition2(start_position, prev_position,prev_direction,
00201                                               *CM, atom_index, step_length, gwNumerics::MaxAbsDistRes);
00202
00203
00204 //   jVector new_position2 = gwNumerics::SetHCPosition(start_position, prev_position,prev_direction,
00205 //                                                     *CM, atom_index, step_length);
00206 //
00207 //   if (euclid(new_position, new_position2) > 0.5) {
00208 //     cout << "\nACK:  atom=" << atom_index << " grd=" << new_position << " trd=" << new_position2 << flush;
00209 //   }
00210
00211     CM->setPosition(atom_index, new_position);
00212
00213 //   if (EC){
00214 //     //    direction = EC->getDirection(new_position);
00215 //     //  cerr << "atom_index:" << atom_index << "pos: " << new_position << "Dir:" << direction << endl;
00216 //     CM->setPosition(atom_index, new_position+direction);
00217 //   }
00218
00219
00220   //  DLList<int> Moved;
00221   //  CM->setPosition(atom_index, prev_position+(prev_direction*step_length));
00222   //  gwNumerics::LocalPert(atom_index, *CM, CM->stress(atom_index), start_position);
00223   //cerr <<"PRE-SPREAD DIST MOVED"<< euclid(prev_position,start_position) << endl;
00224   //  gwNumerics::SpreadR(atom_index, CM->getRadius(atom_index), Moved,
00225   //           0, *CM, CM->getPosition(atom_index), prev_direction);
00226
00227
00228
00229   // shows how much Hard constraint would have moved the atom
00230   // ddd = (ddd -  CM->getMolecule()(atom_index));
00231   // cerr << ddd.mag() << endl;
00232
00233
00234
00235 // #ifdef GNOMAD_OUTPUT
00236 //   printf("  after checks for atom %d\n", atom_index);
00237 //   printf("    x, y, z = %lf  %lf  %lf\n", CM->getMolecule()(atom_index)[0],  CM->getMolecule()(atom_index)[1],
00238 //       CM->getMolecule()(atom_index)[2]);
00239 //   printf("    fv = %lf\n",  CM->stress(atom_index));
00240 //
00241 // //   printf("    atomcount = %d\n", );
00242 // //   maxresid = 0.0;
00243 // //   for(i=0; i<ndis; i++) {
00244 // //     if(dis[i].resid>maxresid) {
00245 // //       maxresid = dis[i].resid;
00246 // //       maxatom1 = dis[i].atom1;
00247 // //       maxatom2 = dis[i].atom2;
00248 // //     }
00249 // //   }
00250
```

4.31.4 Member Data Documentation

4.31.4.1 int gwRelaxAtom::atom_index [private]

The index of the atom in the molecule this process is associated with.

Definition at line 67 of file gw_relax_atom.h.

4.31.4.2 int gwRelaxAtom::max_iterations = 40 [static]

Maximum nuber of times to perform linesearch while updating atom.

Definition at line 32 of file globals.C.

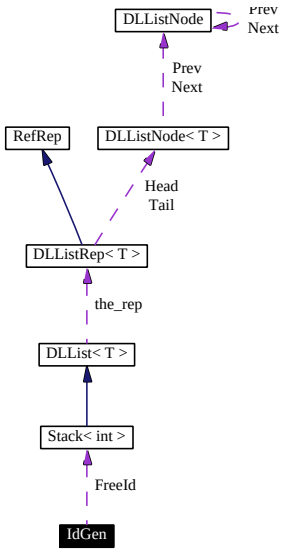
The documentation for this class was generated from the following files:

- **gw_relax_atom.h**
- **globals.C**
- **gw_relax_atom.C**

4.32 IdGen Class Reference

`#include <idgen.h>`

Collaboration diagram for IdGen:



Public Methods

- **IdGen** (int min=0)
- **~IdGen** ()
- int **Max** () const
- int **Peek** () const
- int **Get** ()
- void **Free** (int id)
- void **Reset** ()
- int **NumberIssued** () const
- int **NumberUsed** () const
- int **NumberFree** () const

Private Attributes

- **Stack**<int> **FreeId**
- int **MaxId**
- int **MinId**

Friends

- ostream& **operator**<< (ostream &os, const IdGen &idg)

4.32.1 Constructor & Destructor Documentation

4.32.1.1 IdGen::IdGen (int *min* = 0) [inline]

Definition at line 74 of file idgen.h.

```
00075   { MinId=min; MaxId = MinId; }
00076
```

4.32.1.2 IdGen::~IdGen () [inline]

Definition at line 77 of file idgen.h.

```
00078   { /*empty*/ }
00079
```

4.32.2 Member Function Documentation

4.32.2.1 void IdGen::Free (int *id*) [inline]

Definition at line 103 of file idgen.h.
Referenced by ddGraph::RemVertex().

```
00104   {
00105     // Doesnt check for duplicates in the list...
00106
00107     if(id>=MinId && id<MaxId)
00108         FreeId.Push(id);
00109
```

4.32.2.2 int IdGen::Get () [inline]

Definition at line 90 of file idgen.h.
Referenced by ddGraph::AddVertex().

```
00091   {
00092     int id;
00093     if(FreeId.IsEmpty())
00094       {
00095         id = MaxId;
00096         ++MaxId;
00097       }
00098
```

4.32.2.3 int IdGen::Max () const [inline]

Definition at line 80 of file idgen.h.

```
00081   { return MaxId-1; }
00082
```

4.32.2.4 int IdGen::NumberFree () const [inline]

Definition at line 120 of file idgen.h.
Referenced by NumberUsed().

```
00121   { return FreeId.Number(); }
00122
```

4.32.2.5 int IdGen::NumberIssued () const [inline]

Definition at line 117 of file idgen.h.
Referenced by NumberUsed().

```
00118   { return (MaxId-MinId); }
00119
```

4.32.2.6 int IdGen::NumberUsed () const [inline]

Definition at line 123 of file idgen.h.

```
00124   { return NumberIssued() - NumberFree(); }
00125
```

4.32.2.7 int IdGen::Peek () const [inline]

Definition at line 83 of file idgen.h.
Referenced by ddGraph::NextFreeId().

```
00084   {
00085     if(FreeId.IsEmpty())
00086         return MaxId;
00087     return FreeId.Peek();
00088
```

4.32.2.8 void IdGen::Reset () [inline]

Definition at line 111 of file idgen.h.
Referenced by ddGraph::Clear().

```
00112   {
00113     FreeId.Clear();
00114     MaxId = MinId;
00115
```

4.32.3 Friends And Related Function Documentation

4.32.3.1 ostream& operator<< (ostream & *os*, const IdGen & *idg*) [friend]

Definition at line 126 of file idgen.h.

```
00127   {
00128     os << '[' << idg.MaxId << "] {" << idg.FreeId << '}
00129
```

4.32.4 Member Data Documentation

4.32.4.1 Stack< int > IdGen::FreeId [private]

Definition at line 63 of file idgen.h.

4.32.4.2 int IdGen::MaxId [private]

Definition at line 66 of file idgen.h.

4.32.4.3 int IdGen::MinId [private]

Definition at line 70 of file idgen.h.

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

- **idgen.h**

4.33 IntPriorityQueue Class Template Reference

`#include <ipqueue.h>`

Public Methods

- int **Copy** (const IntPriorityQueue< PT > &ipq)
- **IntPriorityQueue** (int maxsize=**IPQDEFLEN**, int rev=0)
- **IntPriorityQueue** (const IntPriorityQueue< PT > &ipq)
- **~IntPriorityQueue** ()
- void **Build** ()
- int **IsEmpty** () const
- **operator int** () const
- int **IsFull** () const
- int **Size** () const
- int **Number** () const
- int **IsBad** () const
- int **BadId** (int id) const
- int **In** (int id) const
- void **Clear** ()
- int **Add** (int id, const PT &priority)
- PT **Peek** (int id) const
- int **ChangeId** (int oldid, int newid)
- void **Modify** (int id, const PT &newpriority)
- void **RelModify** (int id, const PT &del)
- int **Extreme** () const
- int **Remove** (int &id, PT &priority)
- const IntPriorityQueue<PT>& **operator=** (const IntPriorityQueue< PT > &ipq)
- const PT& **GetPriority** (int id) const
- const PT& **operator[]** (int id) const
- ostream& **PrintTop** (ostream &os, **uint** n) const

Private Methods

- **uint Parent** (**uint** i) const
- **uint Left** (**uint** i) const
- **uint Right** (**uint** i) const
- void **Swap** (**uint** i, **uint** j)
- int **FixDir** (**uint** i) const
- void **Heapify** (**uint** i, **uint** len)
- void **BubbleUp** (**uint** i)

Private Attributes

- int **Rev**
- **uint M**
- **uint N**
- int* **Index**
- int* **Id**
- PT* **Pri**
- int **built**

Friends

- ostream& **operator** (ostream &os, const IntPriorityQueue< PT > &Q)

template<class PT = double> class IntPriorityQueue

4.33.1 Constructor & Destructor Documentation

4.33.1.1 template<class PT> IntPriorityQueue< PT >::IntPriorityQueue<PT> (int *maxsize* = IPQDEFLEN, int *rev* = 0) [inline]

Definition at line 196 of file ipqueue.inc.

```
00197   {
00198     M      = 0;
00199     N      = 0;
00200     Rev    = rev;
00201     built  = 0;
00202
00203     Pri   = 0;
00204     Index = 0;
00205     Id    = 0;
00206
00207     maxsize = (maxsize < 1) ? IPQDEFLEN : maxsize;
00208
00209     if(!(Pri = new PT[maxsize+1]) ||
00210        !(Id  = new int[maxsize+1]) ||
00211        !(Index = new int[maxsize+1]))
00212       {
00213         delete[] Index; Index = 0;
00214         delete[] Id;    Id    = 0;
00215         delete[] Pri;     Pri     = 0;
00216         raise(SIGQUIT);
00217       }
00218
```

4.33.1.2 template<class PT> IntPriorityQueue< PT >::IntPriorityQueue<PT> (const IntPriorityQueue< PT > & *ipq*) [inline]

Definition at line 183 of file ipqueue.inc.

```
00184   {
00185     M     = 0;
00186     N     = 0;
00187     Rev   = 0;
00188     built = 0;
00189     Pri   = 0;
00190     Index = 0;
00191     Id    = 0;
00192     Copy(ipq);
00193
```

4.33.1.3 template<class PT> IntPriorityQueue< PT >::~IntPriorityQueue<PT> () [inline]

Definition at line 223 of file ipqueue.inc.

```
00224   { delete[] Pri; Pri = 0; delete[] Id; Id=0; delete[] Index; Index = 0; }
00225
```

4.33.2 Member Function Documentation

4.33.2.1 template<class PT> int IntPriorityQueue< PT >::Add (int *id*, const PT & *priority*) [inline]

Definition at line 252 of file ipqueue.inc.

Referenced by sdGraph::Dijkstra().

```
00253   {
00254     int i;
00255
00256     // See if the queue's full:
00257     // See if the id's ok:
00258     //
00259     if(IsBad() || BadId(id)) raise(SIGQUIT);
00260     if(IsFull() || In(id)) return 0;
00261
00262     // If the heap has not been built yet, we can just stupidly copy the
00263     // new element into the last spot available in the array.
00264     if(!built)
00265       {
00266         Pri[N+1]      = priority;
00267         Index[id]   = N+1;
00268         Id[N+1]     = id;
00269         // ++N;
00270       }
00271
00272     // Otherwise, we need to insert the new element and maintain the heap
00273     // structure of the array.
00274     else
00275       {
00276         i = N+1;
00277         if(!Rev)
00278           {
00279             while(i>1 && (Pri[Parent(i)] < priority))
00280               {
00281                 if(BadId(i) || BadId(Parent(i))) raise(SIGQUIT);
00282                 Pri[i] = Pri[Parent(i)];
00283                 Id[i] = Id[Parent(i)];
00284                 Index[Id[i]] = i;
00285                 i = Parent(i);
00286               }
00287           }
00288         else
00289           {
00290             while(i>1 && (Pri[Parent(i)] > priority))
00291               {
00292                 if(BadId(i) || BadId(Parent(i))) raise(SIGQUIT);
00293                 Pri[i] = Pri[Parent(i)];
00294                 Id[i] = Id[Parent(i)];
00295                 Index[Id[i]] = i;
00296                 i = Parent(i);
00297               }
00298           }
00299         Pri[i]      = priority;
00300         Id[i]     = id;
00301         Index[id] = i;
00302         //
00303
```

```
00304         /*
00305         ++N;
00306         Pri[N]    = priority;
00307         Id[N]     = id;
00308         Index[id] = N;
00309         BubbleUp(N);
00310         */
00311       }
00312
```

4.33.2.2 template<class PT> int IntPriorityQueue< PT >::BadId (int *id*) const [inline]

Definition at line 19 of file ipqueue.inc.

Referenced by Add(), BubbleUp(), ChangeId(), GetPriority(), In(), Modify(), Peek(), and RelModify().

```
00020   { return (id<0 || id > (int) M); }
00021
```

4.33.2.3 template<class PT> void IntPriorityQueue< PT >::BubbleUp (uint *i*) [inline, private]

Definition at line 84 of file ipqueue.inc.

Referenced by Modify().

```
00085   {
00086     // if(BadId(i)) raise(SIGQUIT);
00087
00088     if(!Rev)
00089       {
00090         while(i>1 && (Pri[i] > Pri[Parent(i)]))
00091           {
00092             if(BadId(i) || BadId(Parent(i)))
00093                 raise(SIGQUIT);
00094             Swap(i,Parent(i));
00095             i = Parent(i);
00096           }
00097       }
00098     else
00099       {
00100         while(i>1 && (Pri[i] < Pri[Parent(i)]))
00101           {
00102             if(BadId(i) || BadId(Parent(i)))
00103                 raise(SIGQUIT);
00104             Swap(i,Parent(i));
00105             i = Parent(i);
00106           }
00107       }
00108
```

4.33.2.4 template<class PT> void IntPriorityQueue< PT >::Build () [inline]

Definition at line 111 of file ipqueue.inc.

Referenced by Remove().

```
00112   {
00113     if(built) return;
00114     if(N>1)
00115         for(uint i = N>>1; i>=1; --i)
00116             Heapify(i,N);
00117     built=1;
00118
```

4.33.2.5 template<class PT> int IntPriorityQueue< PT >::ChangeId (int *oldid*, int *newid*) [inline]

Definition at line 352 of file ipqueue.inc.

```
00353   {
00354     if(IsBad() || BadId(oldid) || BadId(newid)) raise(SIGQUIT);
00355
00356     if(oldid == newid) return 1;
00357     if(!In(oldid) || In(newid))
00358         return 0;
00359
00360     int ndx = Index[oldid];
00361     Index[oldid]  = -1;
00362     Index[newid]  = ndx;
00363     Id[ndx] = newid;
00364     return 1;
00365
```

4.33.2.6 template<class PT> void IntPriorityQueue< PT >::Clear () [inline]

Definition at line 121 of file ipqueue.inc.

Referenced by IntPriorityQueue().

```
00122   {
00123     for(uint i=0; i<=M; ++i)
00124       {
00125         Index[i]  = -1;
00126         Id[i]     = -1;
00127         Pri[i]    = Rev ? INF : -INF;
00128       }
00129
```

4.33.2.7 template<class PT> int IntPriorityQueue< PT >::Copy (const IntPriorityQueue< PT > & *ipq*) [inline]

Definition at line 134 of file ipqueue.inc.

Referenced by IntPriorityQueue().

```
00135   {
00136     if(this == &ipq) return 1;
00137
00138     PT * tmpPri = Pri;
00139     int * tmpIndex  = Index;
00140     int * tmpId     = Id;
00141     if(ipq.M==0)
00142       {
00143         delete[] Pri;   Pri   = 0;
```

```
00144         delete[] Index; Index = 0;
00145         delete[] Id;    Id    = 0;
00146         N = M = 0;
00147         return 1;
00148       }
00149     if(M != ipq.M)
00150       {
00151         if(!(Pri   = new PT[ipq.M+1]) ||
00152            !(Index = new int[ipq.M+1])    ||
00153            !(Id    = new int[ipq.M+1]))
00154           {
00155             delete[] Pri;   Pri   = tmpPri;
00156             delete[] Index; Index = tmpIndex;
00157             delete[] Id;    Id    = tmpId;
00158             return 0;
00159           }
00160         delete[] tmpPri;
00161         delete[] tmpIndex;
00162         delete[] tmpId;
00163         M = ipq.M;
00164       }
00165     N = ipq.N;
00166     Rev = ipq.Rev;
00167     built = ipq.built;
00168     for(uint i=0; i<=M; ++i)
00169       {
00170         Pri[i]   = ipq.Pri[i];
00171         Index[i] = ipq.Index[i];
00172         Id[i]    = ipq.Id[i];
00173       }
00174
```

4.33.2.8 template<class PT> int IntPriorityQueue< PT >::Extreme () const [inline]

Definition at line 400 of file ipqueue.inc.

```
00401   {
00402     if(IsBad() || N<1)
00403         return -1;
00404     return Id[1];
00405
```

4.33.2.9 template<class PT> int IntPriorityQueue< PT >::FixDir (uint *i*) const [inline, private]

Definition at line 317 of file ipqueue.inc.

Referenced by Modify().

```
00318   {
00319     uint l=Left(i), r=Right(i);
00320     if(!Rev)
00321       {
00322         if(i>1 && Pri[i] > Pri[Parent(i)]) return 1;
00323         else
00324           {
00325             if((l<=N) && (Pri[i] < Pri[l]) ||
00326                (r<=N) && (Pri[i] < Pri[r]))
00327                 return -1;
```

```
00328           }
00329       }
00330     else
00331       {
00332         if(i>1 && Pri[i] < Pri[Parent(i)]) return 1;
00333         else
00334           {
00335             if((l<=N) && (Pri[i] > Pri[l]) ||
00336                (r<=N) && (Pri[i] > Pri[r]))
00337                  return -1;
00338           }
00339       }
00340
```

4.33.2.10 template<class PT> const PT & IntPriorityQueue< PT >::GetPriority (int *id*) const [inline]

Definition at line 442 of file ipqueue.inc.

Referenced by operator[]().

```
00443   {
00444     if(IsBad() || BadId(id)) raise(SIGQUIT);
00445     return Pri[Index[id]];
00446
```

4.33.2.11 template<class PT> void IntPriorityQueue< PT >::Heapify (uint *i*, uint *len*) [inline, private]

Definition at line 42 of file ipqueue.inc.

Referenced by Build(), Modify(), and Remove().

```
00043   {
00044     uint l = Left(i), r = Right(i), extreme;
00045
00046     // if(BadId(i) || len>N)
00047     //  raise(SIGQUIT);
00048
00049     if(!Rev)
00050       {
00051         if((l<=len) && (Pri[l] > Pri[i]))
00052           extreme = l;
00053         else
00054           extreme = i;
00055
00056         if((r<=len) && (Pri[r]>Pri[extreme]))
00057           extreme = r;
00058
00059         if(extreme != i)
00060           {
00061             Swap(i,extreme);
00062             Heapify(extreme, len);
00063           }
00064       }
00065     else
00066       {
00067         if((l<=len) && (Pri[l] < Pri[i]))
00068           extreme = l;
00069         else
```

```
00070           extreme = i;
00071
00072         if((r<=len) && (Pri[r]<Pri[extreme]))
00073           extreme = r;
00074
00075         if(extreme != i)
00076           {
00077             Swap(i,extreme);
00078             Heapify(extreme, len);
00079           }
00080       }
00081
```

4.33.2.12 template<class PT> int IntPriorityQueue< PT >::In (int *id*) const [inline]

Definition at line 248 of file ipqueue.inc.

Referenced by Add(), ChangeId(), Modify(), Peek(), and RelModify().

```
00249   { return (BadId(id) ? 0 : (Index[id] != -1)); }
00250
```

4.33.2.13 template<class PT> int IntPriorityQueue< PT >::IsBad () const [inline]

Definition at line 14 of file ipqueue.inc.

Referenced by Add(), ChangeId(), Extreme(), GetPriority(), Modify(), Peek(), PrintTop(), RelModify(), and Remove().

```
00015     { return 0; }
00016
```

4.33.2.14 template<class PT> int IntPriorityQueue< PT >::IsEmpty () const [inline]

Definition at line 228 of file ipqueue.inc.

Referenced by sdGraph::Dijkstra().

```
00229   { return N==0; }
00230
```

4.33.2.15 template<class PT> int IntPriorityQueue< PT >::IsFull () const [inline]

Definition at line 236 of file ipqueue.inc.

Referenced by Add().

```
00237   { return N==M; }
00238
```

4.33.2.16 template<class PT> uint IntPriorityQueue< PT >::Left (uint *i*) const [inline, private]

Definition at line 6 of file ipqueue.inc.

Referenced by FixDir(), and Heapify().

```
00007   { return (i << 1); }
00008
```

4.33.2.17 template<class PT> void IntPriorityQueue< PT >::Modify (int *id*, const PT & *newpriority*) [inline]

Definition at line 368 of file ipqueue.inc.

Referenced by sdGraph::Dijkstra(), and RelModify().

```
00369   {
00370     if(IsBad() || BadId(id)) raise(SIGQUIT);
00371     if(!In(id)) return;
00372
00373     uint ndx = Index[id];
00374     int dir;
00375
00376     if(ndx>N) raise(SIGQUIT);
00377
00378     Pri[ndx] = newpriority;
00379
00380     if(built)
00381       {
00382         if((dir = FixDir(ndx)) == -1)   // Modify downward
00383             Heapify(ndx,N);
00384         else if(dir == 1)               // Modify upward
00385             BubbleUp(ndx);
00386       }
00387
```

4.33.2.18 template<class PT> int IntPriorityQueue< PT >::Number () const [inline]

Definition at line 244 of file ipqueue.inc.

```
00245   { return (int) N; }
00246
```

4.33.2.19 template<class PT> uint IntPriorityQueue< PT >::Parent (uint *i*) const [inline, private]

Definition at line 2 of file ipqueue.inc.

Referenced by Add(), BubbleUp(), and FixDir().

```
00003   { return (i >> 1); }
00004
```

4.33.2.20 template<class PT> PT IntPriorityQueue< PT >::Peek (int *id*) const [inline]

Definition at line 344 of file ipqueue.inc.

```
00345   {
00346     if(IsBad() || BadId(id)) raise(SIGQUIT);
00347     if(!In(id)) return -1.0;
00348     return Pri[ Index[id] ];
00349
```

4.33.2.21 template<class PT> ostream & IntPriorityQueue< PT >::PrintTop (ostream & *os*, uint *n*) const [inline]

Definition at line 466 of file ipqueue.inc.

```
00467   {
00468     if(IsBad())
00469       {
00470         os << "<<BADIPQUEUE>>";
00471         return os;
00472       }
00473
00474     n = (n>N) ? N : n;
00475     for(uint i=1; i<=n; ++i)
00476       {
00477         os << Id[i];
00478         if(i!=n) os << ' ';
00479       }
00480
```

4.33.2.22 template<class PT> void IntPriorityQueue< PT >::RelModify (int *id*, const PT & *del*) [inline]

Definition at line 390 of file ipqueue.inc.

```
00391   {
00392     if(IsBad() || BadId(id)) raise(SIGQUIT);
00393     if(!In(id)) return;
00394     uint ndx = Index[id];
00395     if(ndx>N) raise(SIGQUIT);
00396     Modify(id, Pri[ndx]+del);
00397
```

4.33.2.23 template<class PT> int IntPriorityQueue< PT >::Remove (int & *id*, PT & *priority*) [inline]

Definition at line 408 of file ipqueue.inc.

Referenced by sdGraph::Dijkstra().

```
00409   {
00410     if(IsBad() || N==0)
00411       {
00412         priority = Rev ? INF : -INF;
00413         id = -1;
```

```
00414         return 0;
00415       }
00416
00417     if(!built) Build();
00418
00419     priority = Pri[1];
00420     id       = Id[1];
00421     if(N == 1)
00422       {
00423         Pri[1] = Rev ? INF : -INF;
00424         Id[1]  = -1;
00425         Index[id] = -1;
00426         N = 0;
00427         return 1;
00428       }
00429
```

4.33.2.24 template<class PT> uint IntPriorityQueue< PT >::Right (uint *i*) const [inline, private]

Definition at line 10 of file ipqueue.inc.

Referenced by FixDir(), and Heapify().

```
00011   { return ((i << 1) | ((uint) 1)); }
00012
```

4.33.2.25 template<class PT> int IntPriorityQueue< PT >::Size () const [inline]

Definition at line 240 of file ipqueue.inc.

```
00241   { return (int) M; }
00242
```

4.33.2.26 template<class PT> void IntPriorityQueue< PT >::Swap (uint *i*, uint *j*) [private]

Definition at line 23 of file ipqueue.inc.

Referenced by BubbleUp(), and Heapify().

```
00024   {
00025     // if(IsBad() || BadId(i) || BadId(j) || i>N || j>N) raise(SIGQUIT);
00026     if(i==j) return;
00027
00028     PT d = Pri[i];
00029     Pri[i]   = Pri[j];
00030     Pri[j]   = d;
00031
00032     int k        = Index[Id[i]];
00033     Index[Id[i]] = Index[Id[j]];
00034     Index[Id[j]] = k;
00035
00036     k     = Id[i];
00037     Id[i] = Id[j];
00038     Id[j] = k;
00039
```

4.33.2.27 template<class PT> IntPriorityQueue< PT >::operator int () const [inline]

Definition at line 232 of file ipqueue.inc.

```
00233   { return N>0; }
00234
```

4.33.2.28 template<class PT> const IntPriorityQueue< PT > & IntPriorityQueue< PT >::operator= (const IntPriorityQueue< PT > & *ipq*) [inline]

Definition at line 179 of file ipqueue.inc.

```
00180   { Copy(ipq); return *this; }
00181
```

4.33.2.29 template<class PT> const PT & IntPriorityQueue< PT >::operator[] (int *id*) const [inline]

Definition at line 449 of file ipqueue.inc.

```
00450   { return GetPriority(id); }
00451
```

4.33.3 Friends And Related Function Documentation

4.33.3.1 template<class PT = double> ostream& operator (ostream & *os*, const IntPriorityQueue< PT > & *Q*) [friend]

4.33.4 Member Data Documentation

4.33.4.1 template<class PT = double> int * IntPriorityQueue<PT>::Id [private]

Definition at line 105 of file ipqueue.h.

4.33.4.2 template<class PT = double> int * IntPriorityQueue<PT>::Index [private]

Definition at line 102 of file ipqueue.h.

4.33.4.3 template<class PT = double> uint IntPriorityQueue<PT>::M [private]

Definition at line 96 of file ipqueue.h.

4.33.4.4 template<class PT = double> uint IntPriorityQueue<PT>::N [private]

Definition at line 99 of file ipqueue.h.

4.33.4.5 template<class PT = double> PT * IntPriorityQueue<PT>::Pri [private]

Definition at line 109 of file ipqueue.h.

4.33.4.6 template<class PT = double> int IntPriorityQueue<PT>::Rev [private]

Definition at line 91 of file ipqueue.h.

4.33.4.7 template<class PT = double> int IntPriorityQueue<PT>::built [private]

Definition at line 112 of file ipqueue.h.

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

- **ipqueue.h**
- **ipqueue.inc**

4.34 IntVec Class Reference

`#include <intvec.h>`

Public Methods

- int **AppendUnique** (int newval)
- int **Append** (int newval)
- int **Resize** (int newlen)
- int **Copy** (const IntVec &iv)
- **IntVec** (int len=**CreationLength**)
- **IntVec** (int len, int intital_value)
- **IntVec** (int len, int first_value, int delta_val)
- **IntVec** (const IntVec &iv)
- **~IntVec** ()
- int **Length** () const
- **operator int** () const
- **operator int** * () const
- const IntVec& **operator=** (const IntVec &iv)
- bool **Contains** (int v) const
- const int& **Access** (int i) const
- int& **Access** (int i)
- const int& **operator**[] (int i) const
- int& **operator**[] (int i)
- const IntVec& **set** (int val)
- const IntVec& **setrange** (int val, int delta)
- const IntVec& **zero** ()
- int **dynread** (istream &is, const **DS** &compdelim="\t", const **DS** &vectdelim="\n")
- int **InputBinary** (istream &is)
- int **Input** (istream &is)
- int **OutputBinary** (ostream &os) const
- int **Output** (ostream &os) const
- **MAKE_IO_STREAM_OPERATORS** (IntVec)
- **MAKE_BIN_STREAM_OPERATORS** (IntVec)

Static Public Attributes

- const int **DefCreationLength** = 3
- int **CreationLength** = **DefCreationLength**
- char **IODelimiter** =

Private Attributes

- int* **I**
- int **L**

Friends

- int **operator==** (const IntVec &a, const IntVec &b)
- int **operator!=** (const IntVec &a, const IntVec &b)
- int **operator<=** (const IntVec &a, const IntVec &b)
- int **operator>=** (const IntVec &a, const IntVec &b)
- int **operator<** (const IntVec &a, const IntVec &b)
- int **operator>** (const IntVec &a, const IntVec &b)
- IntVec **cat** (const IntVec &a, const IntVec &b)

4.34.1 Constructor & Destructor Documentation

4.34.1.1 IntVec::IntVec (int *len* = CreationLength) [inline]

Definition at line 77 of file intvec.inc.

```
00078   { I = 0; L = 0; Resize(len); }
00079
```

4.34.1.2 IntVec::IntVec (int *len*, int *initial_value*) [inline]

Definition at line 80 of file intvec.inc.

```
00081   { I = 0; L = 0; Resize(len); set(initial_value); }
00082
```

4.34.1.3 IntVec::IntVec (int *len*, int *first_value*, int *delta*) [inline]

Definition at line 83 of file intvec.inc.

```
00084   { I = 0; L = 0; Resize(len); setrange(first_value, delta); }
00085
```

4.34.1.4 IntVec::IntVec (const IntVec & *iv*) [inline]

Definition at line 86 of file intvec.inc.

```
00087   { I = 0; L = 0; Copy(iv); }
00088
```

4.34.1.5 IntVec::~IntVec () [inline]

Definition at line 89 of file intvec.inc.

```
00090   { delete[] I; }
00091
```

4.34.2 Member Function Documentation

4.34.2.1 int & IntVec::Access (int *i*) [inline]

Definition at line 107 of file intvec.inc.

```
00108   {
00109     if(i<0 || i>=L) { cerr << "IntVec::Access: out of bounds\n"; raise(SIGQUIT); }
00110
```

4.34.2.2 const int & IntVec::Access (int *i*) const [inline]

Definition at line 101 of file intvec.inc.

Referenced by operator[]().

```
00102   {
00103     if(i<0 || i>=L) { cerr << "IntVec::Access const: out of bounds\n"; raise(SIGQUIT); }
00104
```

4.34.2.3 int IntVec::Append (int *newval*) [inline]

Definition at line 27 of file intvec.inc.

Referenced by Residue::AddAtom().

```
00028 {
00029   int * tmp = I;
00030   I = new int[L+1];
00031   memcpy(I,tmp, sizeof(int)*L);
00032   delete[] tmp;
00033   I[L]=newval;
00034   L++;
00035   return 1;
00036
00037
```

4.34.2.4 int IntVec::AppendUnique (int *newval*) [inline]

Definition at line 11 of file intvec.inc.

Referenced by Atom::AddBond(), Constraint::addAtomSupport(), and Constraint::addConstraintSupport().

```
00012 {
00013
00014   for (int i = 0; i < L; i++){
00015     if (I[i] == newval) return 1;
00016   }
00017
```

4.34.2.5 bool IntVec::Contains (int *v*) const [inline]

Definition at line 52 of file intvec.inc.

Referenced by Atom::Bonded().

```
00052                                       {
00053   for (int i = 0; i < L; i++){
00054     if (I[i] == v) return true;
00055   }
00056
```

4.34.2.6 int IntVec::Copy (const IntVec & *iv*) [inline]

Definition at line 59 of file intvec.inc.

Referenced by IntVec(), and operator=().

```
00060   {
00061     if(&iv == this) return 1;
00062     if(!Resize(iv.L))
00063       return 0;
00064     memcpy(I,iv.I, sizeof(int)*L);
00065     return 1;
00066
```

4.34.2.7 int IntVec::Input (istream & *is*) [inline]

Definition at line 178 of file intvec.inc.

```
00179   {
00180     int w;
00181     for(int i=0; i<L; ++i)
00182     {
00183       if(!(is >> w)) return i;
00184       I[i] = w;
00185     }
00186
```

4.34.2.8 int IntVec::InputBinary (istream & *is*) [inline]

Definition at line 199 of file intvec.inc.

```
00200   {
00201     // Read "L":
00202     int e;
00203     is >= e;
00204     if(!Resize(e))
00205         { cerr << "IntVec::InputBinary\n"; raise(SIGQUIT); }
00206
```

4.34.2.9 int IntVec::Length () const [inline]

Definition at line 92 of file intvec.inc.

Referenced by Constraint::Input(), ConstraintManager::add(), Constraint::arity(), Constraint::assertValidity(), ConstraintManager::getAssocAtoms(), and Constraint::notifyRemove().

```
00093   { return L; }
00094
```

4.34.2.10 IntVec::MAKE_BIN_STREAM_OPERATORS (IntVec)

4.34.2.11 IntVec::MAKE_IO_STREAM_OPERATORS (IntVec)

4.34.2.12 int IntVec::Output (ostream & *os*) const [inline]

Definition at line 189 of file intvec.inc.

```
00190   {
00191     for(int i=0; i<L; ++i)
00192     {
00193       os << I[i];
00194       if(i != L-1) os << IntVec::IODelimiter;
00195     }
00196
```

4.34.2.13 int IntVec::OutputBinary (ostream & *os*) const [inline]

Definition at line 215 of file intvec.inc.

```
00216   {
00217     // Write "L":
00218     //
00219     os <= L;
00220
00221     // Write "I":
00222     //
00223     if(I && L>0)
00224       {
00225         os <= 1;
00226         os.write((char*)I,L*sizeof(int));
00227       }
00228
```

4.34.2.14 int IntVec::Resize (int *newlen*) [inline]

Definition at line 39 of file intvec.inc.

Referenced by Atom::Atom(), Copy(), InputBinary(), IntVec(), Residue::Setup(), and dynread().

```
00040   {
00041     if(L == newlen) return 1;
00042     if(newlen<1)
00043       { delete[] I; I=0; L=0; return 1; }
00044     int * tmp = I;
00045     if(!(I = new int[newlen]))
00046       { I=tmp; return 0; }
00047
```

4.34.2.15 int IntVec::dynread (istream & *is*, const DS & *compdelim* = " \t", const DS & *vectdelim* = "\n")

Definition at line 25 of file intvec.C.

```
00027   {
00028     if(is.eof() || is.bad()) return 0;
```

```
00029
00030     DS old_delim = GetDSDelim();
00031     SetDSDelim(vectdelim);
00032
00033     DLList<int> ComponentList;
00034     DS VecString;
00035     int component;
00036
00037     // Read in the next string from the input stream:
00038     is >> VecString;
00039     DSIterator it(VecString,compdelim);
00040
00041     // Extract the components from the string:
00042     for(; it; ++it)
00043       {
00044         component = ($\ast$it).ToInt();
00045         ComponentList.AddToBack(component);
00046       }
00047
```

4.34.2.16 IntVec::operator int () const [inline]

Definition at line 95 of file intvec.inc.

```
00096   { return L; }
00097
```

4.34.2.17 IntVec::operator int * () const

Definition at line 17 of file intvec.C.

```
00018   {
00019     int * i = 0;
00020     if(!I || L<1 || !(i = new int[L])) return 0;
00021     memcpy(i,I,sizeof(int)*L);
00022     return i;
00023
```

4.34.2.18 const IntVec & IntVec::operator= (const IntVec & *iv*) [inline]

Definition at line 98 of file intvec.inc.

```
00099   { Copy(iv); return *this; }
00100
```

4.34.2.19 int & IntVec::operator[] (int *i*) [inline]

Definition at line 116 of file intvec.inc.

```
00117   { return Access(i); }
00118
```

4.34.2.20 const int & IntVec::operator[] (int *i*) const [inline]

Definition at line 113 of file intvec.inc.

```
00114   { return Access(i); }
00115
```

4.34.2.21 const IntVec & IntVec::set (int *val*) [inline]

Definition at line 68 of file intvec.inc.
Referenced by IntVec(), and zero().

```
00069   { for(int i=0; i<L; ++i) I[i] = val; return *this; }
00070
```

4.34.2.22 const IntVec & IntVec::setrange (int *val*, int *delta*) [inline]

Definition at line 71 of file intvec.inc.
Referenced by IntVec().

```
00072   { for(int i=0; i<L; ++i) I[i] = val+(i*delta); return *this; }
00073
```

4.34.2.23 const IntVec & IntVec::zero () [inline]

Definition at line 74 of file intvec.inc.

```
00075   { return set(0); }
00076
```

4.34.3 Friends And Related Function Documentation

4.34.3.1 IntVec cat (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 233 of file intvec.inc.

```
00234   {
00235     IntVec c(a.L+b.L);
00236     for(int i=0; i<a.L; ++i) { c.I[i] = a.I[i]; }
00237     for(int i=0; i<b.L; ++i) { c.I[a.L+i] = b.I[i]; }
00238
```

4.34.3.2 int operator!= (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 127 of file intvec.inc.

```
00128   { return !(a == b); }
00129
```

4.34.3.3 int operator< (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 154 of file intvec.inc.

```
00155   {
00156     if(a.L > b.L) return 0;
00157     if(a.L < b.L) return 1;
00158     for(int i=0; i<a.L; ++i)
00159       {
00160         if(a.I[i] > b.I[i]) return 0;
00161         if(a.I[i] < b.I[i]) return 1;
00162       }
00163
```

4.34.3.4 int operator<= (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 130 of file intvec.inc.

```
00131   {
00132     if(a.L > b.L) return 0;
00133     if(a.L < b.L) return 1;
00134     for(int i=0; i<a.L; ++i)
00135       {
00136         if(a.I[i] > b.I[i]) return 0;
00137         if(a.I[i] < b.I[i]) return 1;
00138       }
00139
```

4.34.3.5 int operator== (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 119 of file intvec.inc.

```
00120   {
00121     if(a.L != b.L) return 0;
00122     for(int i=0; i<a.L; ++i)
00123         if(a.I[i] != b.I[i]) return 0;
00124     return 1;
00125
```

4.34.3.6 int operator> (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 166 of file intvec.inc.

```
00167   {
00168     if(a.L < b.L) return 0;
00169     if(a.L > b.L) return 1;
00170     for(int i=0; i<a.L; ++i)
00171       {
00172         if(a.I[i] < b.I[i]) return 0;
00173         if(a.I[i] > b.I[i]) return 1;
00174       }
00175
```

4.34.3.7 int operator>= (const IntVec & *a*, const IntVec & *b*) [friend]

Definition at line 142 of file intvec.inc.

```
00143   {
00144     if(a.L < b.L) return 0;
00145     if(a.L > b.L) return 1;
00146     for(int i=0; i<a.L; ++i)
00147       {
00148         if(a.I[i] < b.I[i]) return 0;
00149         if(a.I[i] > b.I[i]) return 1;
00150       }
00151
```

4.34.4 Member Data Documentation

4.34.4.1 int IntVec::CreationLength = DefCreationLength [static]

Definition at line 14 of file intvec.C.

4.34.4.2 const int IntVec::DefCreationLength = 3 [static]

Definition at line 13 of file intvec.C.

4.34.4.3 int * IntVec::I [private]

Definition at line 90 of file intvec.h.

4.34.4.4 char IntVec::IODelimiter = [static]

Definition at line 15 of file intvec.C.

4.34.4.5 int IntVec::L [private]

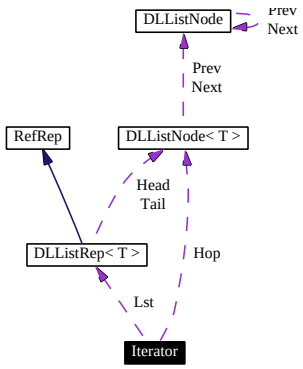
Definition at line 91 of file intvec.h.

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

- **intvec.h**
- **intvec.inc**
- **intvec.C**

4.35 Iterator Class Reference

Collaboration diagram for Iterator:



Public Methods

- void **Reset** ()
- T& **operator** * ()
- **operator int** () const
- const Iterator& **operator++** ()
- void **Copy** (const Iterator &it)
- const Iterator& **operator=** (const Iterator &it)
- **Iterator** ()
- **Iterator** (const Iterator &it)
- **~Iterator** ()

Private Methods

- **Iterator** (**DLList**< T > &dll)

Private Attributes

- **DLListRep**<T>* **Lst**
- **DLListNode**<T>* **Hop**

Friends

- class **DLList**< **T** >

4.35.1 Constructor & Destructor Documentation

4.35.1.1 Iterator::Iterator () [inline]

Definition at line 31 of file dlliter.inc.

```
00031                          { Lst = 0; Hop = 0; }
00032
```

4.35.1.2 Iterator::Iterator (const Iterator & *it*) [inline]

Definition at line 32 of file dlliter.inc.

```
00033                   { Lst = 0; Hop = 0; Copy(it); }
00034
```

4.35.1.3 Iterator::~Iterator () [inline]

Definition at line 34 of file dlliter.inc.

```
00034                             { if(Lst) Lst->Dereference(); Lst=0; }
00035
```

4.35.1.4 Iterator::Iterator (DLList< T > & *dll*) [inline, private]

Definition at line 36 of file dlliter.inc.

```
00037                   {
00038                     Lst = dll.the_rep;
00039                     if(Lst) Lst->Reference();
00040                     Reset();
00041
```

4.35.2 Member Function Documentation

4.35.2.1 void Iterator::Copy (const Iterator & *it*) [inline]

Definition at line 17 of file dlliter.inc.
Referenced by Iterator(), and operator=().

```
00018                   {
00019                     if(Lst != it.Lst)
00020                       {
00021                         if(Lst) Lst->Dereference();
00022                         Lst = it.Lst;
00023                         if(Lst) Lst->Reference();
00024                       }
00025
```

4.35.2.2 void Iterator::Reset () [inline]

Definition at line 12 of file dlliter.inc.
Referenced by Iterator().

```
00012                             { Hop = Lst->Head; }
00013
```

4.35.2.3 T & Iterator::operator * () [inline]

Definition at line 13 of file dlliter.inc.

```
00013                                 { return Hop->Data; }
00014
```

4.35.2.4 Iterator::operator int () const [inline]

Definition at line 14 of file dlliter.inc.

```
00014                                   { return Hop!=0; }
00015
```

4.35.2.5 const Iterator & Iterator::operator++ () [inline]

Definition at line 15 of file dlliter.inc.

```
00016                   { if(Hop) Hop = Hop->Next; return *this; }
00017
```

4.35.2.6 const Iterator & Iterator::operator= (const Iterator & *it*) [inline]

Definition at line 28 of file dlliter.inc.

```
00029                   { Copy(it); return *this; }
00030
```

4.35.3 Friends And Related Function Documentation

4.35.3.1 class DLList [friend]

Definition at line 10 of file dlliter.inc.

4.35.4 Member Data Documentation

4.35.4.1 DLListNode< T > * Iterator::Hop [private]

Definition at line 43 of file dlliter.inc.

4.35.4.2 DLListRep< T > * Iterator::Lst [private]

Definition at line 42 of file dlliter.inc.

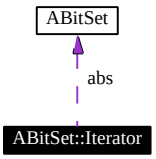
The documentation for this class was generated from the following file:

- **dlliter.inc**

4.36 ABitSet::Iterator Class Reference

`#include <abitset.h>`

Collaboration diagram for ABitSet::Iterator:



Public Methods

- **Iterator** (const **ABitSet** &iabs)
- **~Iterator** ()
- void **Reset** ()
- bool **operator++** ()
- bool **operator–** ()
- **uint operator** * () const
- **operator bool** () const

Protected Methods

- void **find_prev** ()
- void **find_next** ()

Private Attributes

- **uint i**
- const **ABitSet**& **abs**

4.36.1 Constructor & Destructor Documentation

4.36.1.1 ABitSet::Iterator::Iterator (const ABitSet & *iabs*)

Definition at line 376 of file abitset.C.

```
00377 : i(0)
00378 , abs(iabs)
00379 {
00380         find_next();
00381
```

4.36.1.2 ABitSet::Iterator::~Iterator ()

Definition at line 383 of file abitset.C.

```
00384 {
00385
```

4.36.2 Member Function Documentation

4.36.2.1 void ABitSet::Iterator::Reset ()

Definition at line 388 of file abitset.C.

```
00389 {
00390         i = 0;
00391         find_next();
00392
```

4.36.2.2 void ABitSet::Iterator::find_next () [protected]

Definition at line 422 of file abitset.C.
Referenced by Iterator(), Reset(), and operator++().

```
00423 {
00424         while ( (i<abs.num) && (!abs[i]) )
00425                 ++i;
00426
```

4.36.2.3 void ABitSet::Iterator::find_prev () [protected]

Definition at line 429 of file abitset.C.
Referenced by operator–().

```
00430 {
00431         while ( (i<abs.num) && (!abs[i]) )
00432                 --i;
00433
```

4.36.2.4 uint ABitSet::Iterator::operator * () const

Definition at line 400 of file abitset.C.

```
00401 {
00402         return i;
00403
```

4.36.2.5 ABitSet::Iterator::operator bool () const

Definition at line 394 of file abitset.C.

```
00395 {
00396         return (i < abs.num);
00397
```

4.36.2.6 bool ABitSet::Iterator::operator++ ()

Definition at line 414 of file abitset.C.

```
00415 {
00416         ++i;
00417         find_next();
00418         return i < abs.num;
00419
```

4.36.2.7 bool ABitSet::Iterator::operator– ()

Definition at line 406 of file abitset.C.

```
00407 {
00408         --i;
00409         find_prev();
00410         return i < abs.num;
00411
```

4.36.3 Member Data Documentation

4.36.3.1 const ABitSet & ABitSet::Iterator::abs [private]

Definition at line 33 of file abitset.h.

4.36.3.2 uint ABitSet::Iterator::i [private]

Definition at line 32 of file abitset.h.

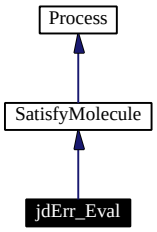
The documentation for this class was generated from the following files:

- **abitset.h**
- **abitset.C**

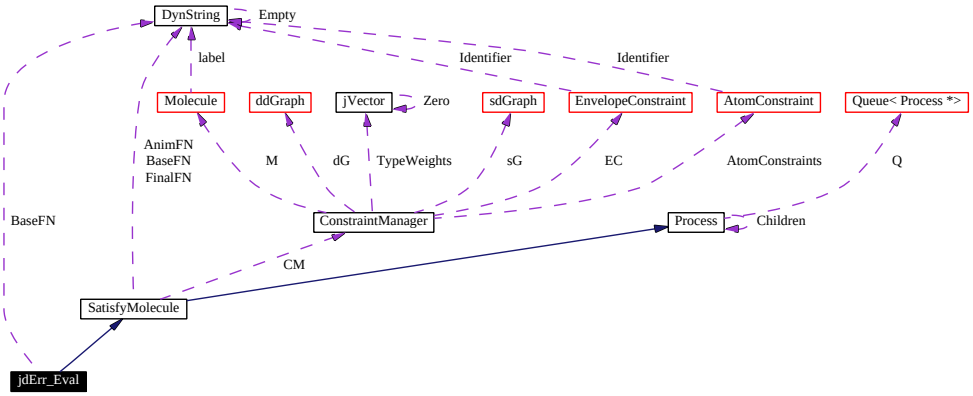
4.37 jdErr_Eval Class Reference

`#include <jd_err_eval.h>`

Inheritance diagram for jdErr_Eval



Collaboration diagram for jdErr_Eval:



Public Methods

- **jdErr_Eval** (**ConstraintManager** &**CM**, ifstream *inStream, int skip_in)
- **~jdErr_Eval** ()
- int **run** ()

Public Attributes

- int **skip**
- **DynString BaseFN**

Private Attributes

- ifstream* **mol_file**

4.37.1 Detailed Description

//

File: **jd_err_eval.h**

Created by: Jon Dugan

jdErr_Eval job.

//

Definition at line 25 of file jd_err_eval.h.

4.37.2 Constructor & Destructor Documentation

4.37.2.1 jdErr_Eval::jdErr_Eval (ConstraintManager & *cm*, ifstream * *inStream*, int *skip_in* = 1)

Definition at line 3 of file jd_err_eval.C.

```
00004   : SatisfyMolecule(cm)
00005 {
00006   mol_file = inStream;
00007   skip =skip_in;
00008   if (skip < 1)
00009     DIE ("invalid slip number, use int > 0");
00010
```

4.37.2.2 jdErr_Eval::~jdErr_Eval ()

Definition at line 12 of file jd_err_eval.C.

```
00013   {
00014
```

4.37.3 Member Function Documentation

4.37.3.1 int jdErr_Eval::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 16 of file jd_err_eval.C.

Referenced by main().

```
00016                   {
00017
00018   DynString FN = "err_out.xmol";
00019   AnimFrame::Enabled =true;
00020   AnimFrame *Out = new AnimFrame(*CM, FN);
00021   jVector E;
00022
00023 #ifdef JOB_VERBOSE
00024   cerr << "Start: \tErr_Eval::run()" << endl;
00025 #endif
00026
00027
00028   int a1 = 0;
00029   int a2 = 0;
00030   int num = CM->numAtoms();
00031   double d;
00032   double msd;
```

```
00033   int numerr = 0;
00034
00035   cerr << "Checking msd: " ;
00036
00037   for (a1 = 0;a1<num;a1++){
00038     for (a2 = a1+1;a2<num;a2++){
00039       d = euclid (CM->getPosition(a1),CM->getPosition(a2));
00040       msd = CM->getMolecule().MSD(a1, a2);
00041       if ( d < msd && ((d-msd)/msd > 1e-3)){
00042         numerr ++;
00043         cerr << endl << endl  << "atom1 = " << a1 << " "<< CM->getAtom(a1).Label() <<endl
00044              << "atom2 = " << a2 << " " << CM->getAtom(a2).Label() <<endl
00045              << "bonded = " << CM->getAtom(a1).Bonded(a2) << endl
00046              << "d = " << d <<endl
00047              << "msd = " << msd;
00048       }
00049     }
00050   }
00051   if (numerr)
00052     cerr << endl << "Found " << numerr << " error(s) with msd" << endl;
00053   else
00054     cerr << "ok" << endl << endl;
00055
00056   if (CM->numConstraints() == 0){
00057     return 0;}
00058
00059   int  i = 0;
00060   cerr << "Collecting constraint data..." << endl;
00061   E =  CM->getStressList();
00062   //  cerr << E << endl;
00063   cerr << i << " Total Error: " << E.sum() << endl;
00064   cerr << i << " Max   Error: " << E.max() << endl;
00065
00066 //   for (int i=0;i<CM->numAtoms();i++){
00067 //     if (E[i] > 0.1){
00068 //       cerr << CM->getAtom(i).Label() << " " << CM->getPosition(i) << " "
00069 //         << E[i] << " " << CM->getEnvelope()->getGradient(CM->getPosition(i)) << endl;
00070 //     }
00071 //   }
00072
00073   cerr << "======================================================================" << endl;
00074
00075   mol_file->ipfx(); // gobbles whitespace & sets flags for good()
00076
00077   cerr << "Writing xmol frame to " << FN << endl;
00078   Out->run();
00079
00080   while (mol_file->good()){
00081     i++;
00082
00083     //    M->writeXmol(cerr);
00084     //    CM->InvalidateAtoms();
00085
00086     if (i%skip == 0){
00087       CM->readNewMoleculeXmol(*mol_file);
00088       CM->InvalidateEnvelope();
00089       E =  CM->getStressList();
00090       cerr << i << " Error: " << E.sum() << endl
00091            << "======================================================================" << endl;
00092
00093       Out->run();
00094     } else {
00095       //CM->readNewMoleculeXmol(*mol_file);
00096       Molecule::skipXmol(*mol_file);
00097     }
```

```
00098       mol_file->ipfx();
00099
00100   }
00101
00102   delete Out;
00103
00104
00105   if (EnvelopeConstraint * E_ptr = CM->getEnvelope()){
00106     DynString F;
00107     char * s;
00108     ofstream * os;
00109     F = "err_out.iv";
00110     s = &(F[0]);  // ACK  not good, but it doesn't leak. -jd
00111     os = new ofstream (s);
00112     E_ptr->WriteVRML_OrigData(*os);
00113     delete os;
00114     cerr << "Wrote envelope data in inventor format to: " << F << endl;
00115
00116     F = "err_out.csx";
00117     s = &(F[0]);  // ACK  not good, but it doesn't leak. -jd
00118     os = new ofstream (s);
00119     E_ptr->E[0]->WriteAvsTxt(*os);
00120     delete os;
00121     cerr << "Wrote envelope as constraint to: " << F << endl;
00122   }
00123
```

4.37.4 Member Data Documentation

4.37.4.1 DynString jdErr_Eval::BaseFN

Reimplemented from **SatisfyMolecule** (p. 440).

Definition at line 34 of file jd_err_eval.h.

4.37.4.2 ifstream * jdErr_Eval::mol_file [private]

Definition at line 39 of file jd_err_eval.h.

4.37.4.3 int jdErr_Eval::skip

Definition at line 32 of file jd_err_eval.h.

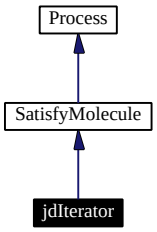
The documentation for this class was generated from the following files:

- **jd_err_eval.h**
- **jd_err_eval.C**

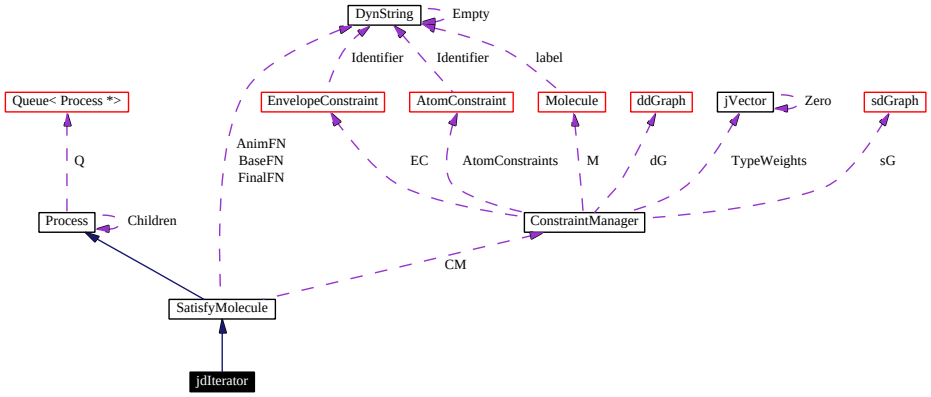
4.38 jdIterator Class Reference

`#include <jd_iterator.h>`

Inheritance diagram for jdIterator



Collaboration diagram for jdIterator:



Public Methods

- **jdIterator** (const jdIterator &ca)
- **jdIterator** (**ConstraintManager** &**CM**)
- **~jdIterator** ()
- int **run** ()

Static Public Attributes

- int **MaxIterations** = 2999

Private Attributes

- int **IterationNumber**

4.38.1 Detailed Description

//

File: **jd_iterator.h**

Created by: Jon Dugan

jdIterator job. iterates over all atoms based on error

//

Definition at line 28 of file jd_iterator.h.

4.38.2 Constructor & Destructor Documentation

4.38.2.1 jdIterator::jdIterator (const jdIterator & *ca*)

Definition at line 3 of file jd_iterator.C.

```
00004   : SatisfyMolecule(*(ca.CM))
00005 {
00006   DIE("jdIterator copy constructor called. ");
00007
```

4.38.2.2 jdIterator::jdIterator (ConstraintManager & *cm*)

Definition at line 10 of file jd_iterator.C.

```
00011   : SatisfyMolecule(cm)
00012 {
00013
00014
```

4.38.2.3 jdIterator::~jdIterator ()

Definition at line 16 of file jd_iterator.C.

```
00017   {
00018     // Don't delete my children (since Process takes care of it).
00019     // i.e. dont do for(int i=0; i<numChildren; ++i) delete Children[i].
00020     // or delete[] Children.
00021
00022     // NB in THIS class I am not adding my children to the queue for
00023     // the parent class, Procss'  As a result, the children need to be
00024     // explicitly deleted.
00025
00026     delete Children[0];
00027     delete Children[1];
00028     delete Children[2];
00029     delete Children[3];
00030     // Normally a job would NOT do this --
00031     // ~Process would do it, but I am not adding these to the queue
00032
```

4.38.3 Member Function Documentation

4.38.3.1 int jdIterator::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 34 of file jd_iterator.C.

```
00034                   {
00035
00036 #ifdef JOB_VERBOSE
00037   cerr << "Start: \tIterator::run()" << endl;
00038 #endif
00039
00040   //CM->PrintGraph(cout);
00041
00042   int StopFlag = 0;
00043   int iteration = 0;
00044
00045   Children = new Process*[4];
00046
00047   Children[0] = new CycleAll(*CM);
00048   Children[1] = new AnimFrame(*CM, AnimFN);
00049   Children[2] = new StopEval(*CM);
00050   Children[3] = new RelaxSubset(*CM, .4);
00051
00052
00053   while (!StopFlag && iteration < MaxIterations){
00054
00055     // CYCLE and RELAX
00056     Children[0]->run();
00057
00058     // RESET CYCLE ATOMS
00059     Children[0]->reset();
00060
00061
00062     //Children[3]->run();
00063     //Children[3]->run();
00064
00065
00066     // WRITE ANIMATION FRAME
00067     Children[1]->run();
00068
00069     // EVALUATE STOP CONDITION
00070     StopFlag = Children[2]->run();  //editing this still -- performs evaluations and stop criteria
00071
00072     iteration++;
00073
00074   }
00075
```

4.38.4 Member Data Documentation

4.38.4.1 int jdIterator::IterationNumber [private]

Definition at line 45 of file jd_iterator.h.

4.38.4.2 int jdIterator::MaxIterations = 2999 [static]

Definition at line 51 of file globals.C.

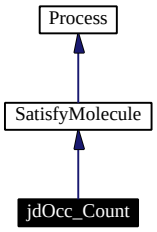
The documentation for this class was generated from the following files:

- **jd_iterator.h**
- **globals.C**
- **jd_iterator.C**

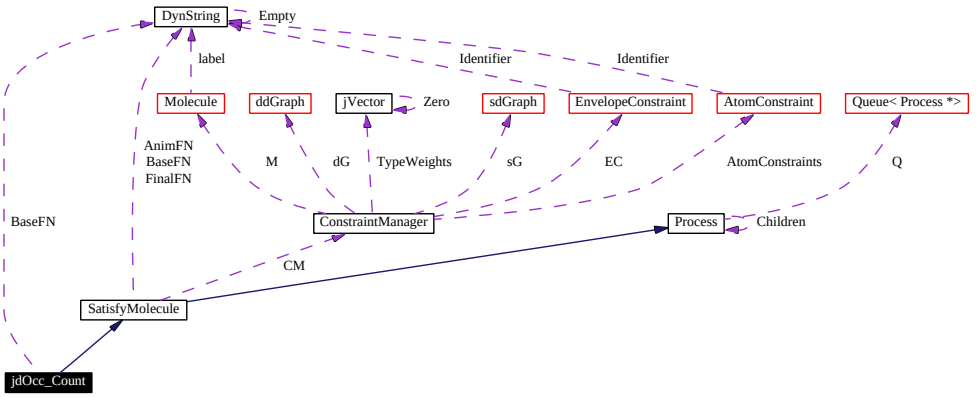
4.39 jdOcc_Count Class Reference

`#include <jd_occ_count.h>`

Inheritance diagram for jdOcc_Count



Collaboration diagram for jdOcc_Count:



Public Methods

- **jdOcc_Count** (**ConstraintManager** &**CM**)
- ~**jdOcc_Count** ()
- int **run** ()

Public Attributes

- **DynString BaseFN**

Static Public Attributes

- double **cube_width** = 3.7

4.39.1 Detailed Description

//

File: **jd_occ_count.h**

Created by: Jon Dugan

jdOcc_Count job.

//

Definition at line 28 of file jd_occ_count.h.

4.39.2 Constructor & Destructor Documentation

4.39.2.1 jdOcc_Count::jdOcc_Count (ConstraintManager & *cm*)

Definition at line 3 of file jd_occ_count.C.

```
00004   : SatisfyMolecule(cm)
00005 {
00006
```

4.39.2.2 jdOcc_Count::~jdOcc_Count ()

Definition at line 8 of file jd_occ_count.C.

```
00009   {
00010
```

4.39.3 Member Function Documentation

4.39.3.1 int jdOcc_Count::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 12 of file jd_occ_count.C.

Referenced by main().

```
00012                     {
00013
00014 #ifdef JOB_VERBOSE
00015   cerr << "Start: \tOcc_Count::run() width =" << cube_width << endl;
00016 #endif
00017
00018   Envelope E;
00019   E.Build(CM->getMolecule(),cube_width);
00020
00021   DynString F;
00022   char * s;
00023   ofstream * os;
00024   F = BaseFN + ".csx";
00025   s = &(F[0]);  // ACK  not good, but it doesn't leak. -jd
00026   os = new ofstream (s);
00027   E.WriteAvsTxt(*os);
00028   delete os;
00029     cerr << "Wrote: " << F << endl;
00030
00031   F = BaseFN + ".iv";
00032   s = &(F[0]);  // ACK  not good, but it doesn't leak. -jd
00033   os = new ofstream (s);
```

```
00034   E.WriteVRML(*os);
00035   delete os;
00036   cerr << "Wrote: " << F << endl;
00037
00038
00039   F = BaseFN + ".density";
00040   s = &(F[0]);  // ACK  not good, but it doesn't leak. -jd
00041   os = new ofstream (s);
00042   E.WriteDensity(*os, CM->getMolecule());
00043   delete os;
00044   cerr << "Wrote: " << F << endl;
00045
00046
00047   //  E.Output(cout);
00048
00049
00050
00051 #ifdef JOB_VERBOSE
00052     cerr << "Ending:\tOcc_Count::run()" << endl;
00053 #endif
00054
00055     return 0; //Normal completion
00056
```

4.39.4 Member Data Documentation

4.39.4.1 DynString jdOcc_Count::BaseFN

Reimplemented from **SatisfyMolecule** (p. 440).

Definition at line 37 of file jd_occ_count.h.

4.39.4.2 double jdOcc_Count::cube_width = 3.7 [static]

Definition at line 62 of file globals.C.

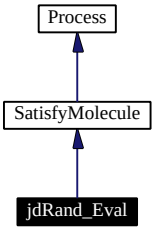
The documentation for this class was generated from the following files:

- **jd_occ_count.h**
- **globals.C**
- **jd_occ_count.C**

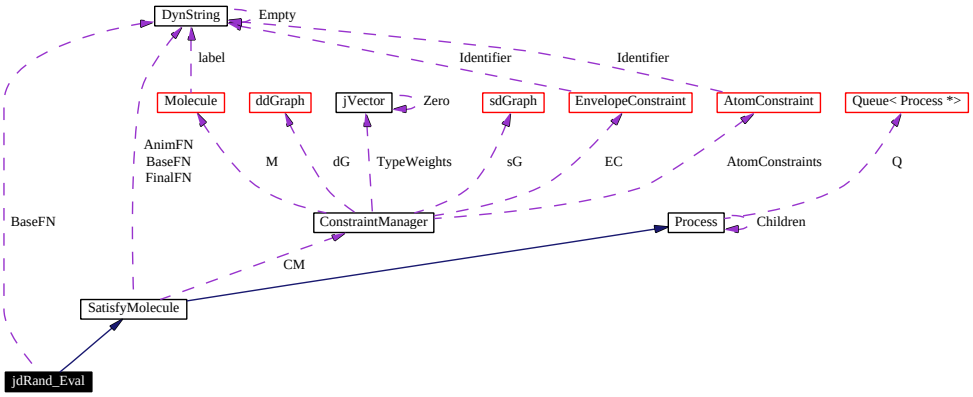
4.40 jdRand_Eval Class Reference

`#include <jd_rand_eval.h>`

Inheritance diagram for jdRand_Eval



Collaboration diagram for jdRand_Eval:



Public Methods

- **jdRand_Eval** (**ConstraintManager** &**CM**, ifstream *inStream, int cycle_in, double pert_in)
- **~jdRand_Eval** ()
- int **run** ()

Public Attributes

- int **cycles**
- double **pert**
- **DynString BaseFN**

Private Attributes

- ifstream* **mol_file**

4.40.1 Detailed Description

//

File: **jd_rand_eval.h**

Created by: Jon Dugan

jdRand_Eval job. perturb a molecule randomly and report on the satisfaction against a set of input constraints.

//

Definition at line 26 of file jd_rand_eval.h.

4.40.2 Constructor & Destructor Documentation

4.40.2.1 jdRand_Eval::jdRand_Eval (ConstraintManager & *cm*, ifstream * *inStream*, int *c_in* = 10, double *p_in* = 0.05)

Definition at line 3 of file jd_rand_eval.C.

```
00005   : SatisfyMolecule(cm)
00006 {
00007   mol_file = inStream;
00008   cycles = c_in;
00009   pert = p_in;
00010
```

4.40.2.2 jdRand_Eval::~jdRand_Eval ()

Definition at line 12 of file jd_rand_eval.C.

```
00013   {
00014
```

4.40.3 Member Function Documentation

4.40.3.1 int jdRand_Eval::run () [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 16 of file jd_rand_eval.C.

Referenced by main().

```
00016                     {
00017
00018   DynString FN = "rand_out.xmol";
00019   AnimFrame::Enabled =true;
00020   AnimFrame *Out = new AnimFrame(*CM, FN);
00021   jVector E;
00022
00023 #ifdef JOB_VERBOSE
00024   cerr << "Start: \tRand_Eval::run()" << endl;
00025 #endif
00026   int  i = 0;
```

```
00027   E =  CM->getStressList();
00028
00029   // REPORT
00030   cerr << "Frame: " << i << " Cycle: " << "original" << " Error: " << E.sum() << endl
00031        << "========================================================================" << endl;
00032   Out->run();
00033   for (int k = 1; k<=cycles;k++){
00034     CM->RandomPerturb(pert);
00035     CM->InvalidateEnvelope();
00036     E =  CM->getStressList();
00037     cerr << "Frame: " << i << " Cycle: " << k << " Error: " << E.sum() << endl
00038          << "========================================================================" << endl;
00039   Out->run();
00040   }
00041
00042
00043   mol_file->ipfx(); // gobbles whitespace & sets flags for good()
00044
00045
00046   while (mol_file->good()){
00047     i++;
00048
00049     //    M->writeXmol(cerr);
00050     //    CM->InvalidateAtoms();
00051
00052
00053       CM->readNewMoleculeXmol(*mol_file);
00054       CM->InvalidateEnvelope();
00055       E =  CM->getStressList();
00056
00057       // REPORT
00058       cerr << "Frame: " << i << " Cycle: " << "original" << " Error: " << E.sum() << endl
00059            << "========================================================================" << endl;
00060       Out->run();
00061       for (int k = 1; k<=cycles;k++){
00062         CM->RandomPerturb(pert);
00063         CM->InvalidateEnvelope();
00064         E =  CM->getStressList();
00065         cerr << "Frame: " << i << " Cycle: " << k << " Error: " << E.sum() << endl
00066              << "========================================================================" << endl;
00067       Out->run();
00068       }
00069
00070       mol_file->ipfx(); //make sure input xmol file is still good
00071
00072   }
00073
```

4.40.4 Member Data Documentation

4.40.4.1 DynString jdRand_Eval::BaseFN

Reimplemented from **SatisfyMolecule** (p. 440).

Definition at line 36 of file jd_rand_eval.h.

4.40.4.2 int jdRand_Eval::cycles

Definition at line 33 of file jd_rand_eval.h.

4.40.4.3 ifstream * jdRand_Eval::mol_file [private]

Definition at line 40 of file jd_rand_eval.h.

4.40.4.4 double jdRand_Eval::pert

Definition at line 34 of file jd_rand_eval.h.

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

- **jd_rand_eval.h**
- **jd_rand_eval.C**

4.41 jVector Class Reference

`#include <jvector.h>`

Collaboration diagram for jVector:

Public Methods

- int **Resize** (int size)
- int **Copy** (const jVector &y)
- **operator double** * () const
- const double& **get** (int i) const
- void **set** (int i, const double &d)
- const jVector& **set** (const double &d)
- const jVector& **zero** ()
- **jVector** (int length=**CreationLength**)
- **jVector** (const jVector &v)
- **jVector** (int length, double d)
- **~jVector** ()
- const jVector& **transpose** ()
- int **Vertical** ()
- int **Length** () const
- double **sum** () const

  Returns the sum of all the components.

- double **sumsqr** () const

  Returns the sum of the squares.

- double **variance** () const

  Returns the variance.

- double **max** () const
- int **argmax** () const
- double **min** () const
- int **argmin** () const
- double **mean** () const
- double **median** () const
- double **nrm** () const

  Returns the norm (square of magnitude.

- jVector **norm_unit** () const

  Returns a jVector of unit length;.

- double **mag** () const

  returns magnitude.

- jVector **unit** () const

Returns a jVector of unit length;.

- const double& **operator**[ ] (int i) const

  indexing.

- double& **operator**[ ] (int i)
- jVector **operator-** ()

  negation.

- const jVector& **operator=** (const jVector &y)
- const jVector& **operator=** (const double &d)
- int **Compare** (const jVector &v) const
- const jVector& **operator-=** (const jVector &y)
- const jVector& **operator+=** (const jVector &y)
- const jVector& **operator%=** (const jVector &y)
- const jVector& **operator** *= (double d)
- const jVector& **operator/=** (double d)
- const jVector& **operator%=** (double d)
- int **Input** (istream &is)
- int **Output** (ostream &os) const
- int **dynread** (istream &is, const **DS** &compdelim="\t", const **DS** &vectdelim="\n")
- int **InputBinary** (istream &is)
- int **OutputBinary** (ostream &os) const
- **MAKE_IO_STREAM_OPERATORS** (jVector)
- **MAKE_BIN_STREAM_OPERATORS** (jVector)

Static Public Attributes

- int **CreationLength** = **DEFAULTLEN**
- char **IODelimiter** =
- jVector **Zero**

Private Methods

- void **mult** (double d)

Private Attributes

- double* **V**
- int **L**
- **usint vert**

Friends

- int **operator==** (const jVector &x, const jVector &y)

  Comparison operators.

- int **operator!=** (const jVector &x, const jVector &y)
- int **operator<** (const jVector &x, const jVector &y)

- int **operator>** (const jVector &x, const jVector &y)
- int **operator<=** (const jVector &x, const jVector &y)
- int **operator>=** (const jVector &x, const jVector &y)
- jVector **operator-** (jVector x, const jVector &y)
- jVector **operator+** (jVector x, const jVector &y)
- double **operator** * (jVector x, const jVector &y)
- jVector **operator%** (jVector x, const jVector &y)
- jVector **operator** * (jVector y, double d)
- jVector **operator** * (double d, jVector y)
- jVector **operator/** (jVector y, double d)
- jVector **operator%** (jVector x, double d)
- double **JVAngle** (const jVector &x, const jVector &y)
- double **euclid** (const jVector &x, const jVector &y)
- ostream& **VBPrintLine** (ostream &os, const jVector &x, const jVector &y, const **DynString** &color)
- ostream& **VBPrintPoint** (ostream &os, const jVector &x, const **DynString** &color)
- ostream& **VBPrintCircle** (ostream &os, const jVector &x, const double &radius, const **DynString** &color)
- void **JVqsort** (jVector &v)
- int **binsearch** (const jVector &v, double d)
- double **Covariance** (const jVector &x, const jVector &y)

4.41.1 Constructor & Destructor Documentation

4.41.1.1 jVector::jVector (int *length* = CreationLength) [inline]

Definition at line 72 of file jvector.h.

```
00073   { L=0; V=0; vert=1; Resize(length); }
00074
```

4.41.1.2 jVector::jVector (const jVector & *v*) [inline]

Definition at line 75 of file jvector.h.

```
00076   { L=0; V=0; Copy(v); }
00077
```

4.41.1.3 jVector::jVector (int *length*, double *d*) [inline]

Definition at line 77 of file jvector.h.

```
00078   { L=0; V=0; Resize(length); set(d); }
00079
```

4.41.1.4 jVector::~jVector () [inline]

Definition at line 80 of file jvector.h.

```
00080             { delete[] V; }
00081
```

4.41.2 Member Function Documentation

4.41.2.1 int jVector::Compare (const jVector & *v*) const [inline]

Definition at line 203 of file jvector.h.

```
00204   {
00205     if(*this==v)
00206       return 0;
00207     if(*this>v)
00208       return 1;
00209     return -1;
00210
```

4.41.2.2 int jVector::Copy (const jVector & *y*) [inline]

Definition at line 48 of file jvector.h.
Referenced by jVector(), and operator=().

```
00049   {
00050     if(&y == this) return 1;
00051     if(!Resize(y.L)) return 0;
00052     memcpy(V, y.V, sizeof(double)*L);
00053     vert = y.vert;
00054     return 1;
00055
```

4.41.2.3 int jVector::Input (istream & *is*)

Definition at line 165 of file jvector.C.

```
00166 {
00167   double w;
00168   for(int i=0; i<L; ++i)
00169   {
00170     if(!(is >> w)) return i;
00171     V[i] = w;
00172   }
00173
```

4.41.2.4 int jVector::InputBinary (istream & *is*)

Definition at line 281 of file jvector.C.

```
00282   {
00283     // Read L:
00284     //
00285     int e;
00286     is >= e;
00287     if(!Resize(e))
00288         { cerr << "jVector::InputBinary\n"; raise(SIGQUIT); }
00289
```

4.41.2.5 int jVector::Length () const [inline]

Definition at line 84 of file jvector.h.

Referenced by StopEval::check_stress_max_tot(), and AnimFrame::run().

```
00084                     { return L; }
00085
```

4.41.2.6 jVector::MAKE_BIN_STREAM_OPERATORS (jVector)

4.41.2.7 jVector::MAKE_IO_STREAM_OPERATORS (jVector)

4.41.2.8 int jVector::Output (ostream & *os*) const

Definition at line 176 of file jvector.C.

```
00177 {
00178   for(int i=0; i<L; ++i)
00179   {
00180     os << V[i];
00181     if(i != L-1) os << IODelimiter;
00182   }
00183
```

4.41.2.9 int jVector::OutputBinary (ostream & *os*) const

Definition at line 304 of file jvector.C.

```
00305   {
00306     // Write "L":
00307     //
00308     os <= L;
00309
00310     // Write "vert":
00311     //
00312     os <= vert;
00313
00314     // Write "V":
00315     //
00316     if(V && L>0)
00317       {
00318         os <= 1;
00319         os.write((char*)V,L*sizeof(double));
00320       }
00321
```

4.41.2.10 int jVector::Resize (int *size*) [inline]

Definition at line 34 of file jvector.h.

Referenced by Copy(), sdGraph::Dijkstra(), Grid::Grid(), InputBinary(), Grid::ReSize(), dynread(), and jVector().

```
00035   {
00036     if(L == size) return 1;
```

```
00037     if(size<1)
00038       { delete[] V; V = 0; L = 0; return 1; }
00039     double * tmp = V;
00040     if(!(V = new double[size]))
00041       { V = tmp; return 0; }
00042
```

4.41.2.11 int jVector::Vertical () [inline]

Definition at line 83 of file jvector.h.

```
00083                 { return (int) vert; }
00084
```

4.41.2.12 int jVector::argmax () const

Definition at line 64 of file jvector.C.

```
00065 {
00066   if(!L) return -1;
00067   double m = V[0];
00068   int mi = 0;
00069   for(int i=1; i<L; ++i) { if(m < V[i]) m = V[i]; mi = i; }
00070
```

4.41.2.13 int jVector::argmin () const

Definition at line 81 of file jvector.C.

```
00082 {
00083   if(!L) return -1;
00084   double m = V[0];
00085   int mi = 0;
00086   for(int i=1; i<L; ++i) { if(m > V[i]) m = V[i]; mi = i; }
00087
```

4.41.2.14 int jVector::dynread (istream & *is*, const DS & *compdelim* = " \t", const DS & *vectdelim* = "\n")

Definition at line 186 of file jvector.C.

Referenced by Molecule::DynRead().

```
00188   {
00189     if(is.eof() || is.bad()) return 0;
00190
00191     DS old_delim = GetDSDelim();
00192     SetDSDelim(vectdelim);
00193
00194     DLList<double> ComponentList;
00195     DS VecString;
00196     // DSIterator it;
00197     double component;
00198
00199     // Read in the next string from the input stream:
```

```
00200     is >> VecString;
00201     // cout << '[' << VecString << "]\n";
00202     // it.SetString(VecString);
00203     // it.SetDelim(compdelim);
00204     DSIterator it(VecString,compdelim);
00205
00206     // Extract the components from the string:
00207     for(; it; ++it)
00208       {
00209         component = ($\ast$it).ToDbl();
00210         ComponentList.AddToBack(component);
00211       }
00212
```

4.41.2.15 const double & jVector::get (int *i*) const [inline]

Definition at line 60 of file jvector.h.

```
00061   { return V[i]; }
00062
```

4.41.2.16 double jVector::mag () const

returns magnitude.

Definition at line 110 of file jvector.C.

Referenced by EnvelopeConstraint::FindModelTranslationToCOM(), JVAngle(), EnvelopeConstraint::MoveModelbyGrad(), Envelope::WriteVRMLgr(), gwRelaxAtom::run(), and unit().

```
00111   { return sqrt(nrm()); }
00112
```

4.41.2.17 double jVector::max () const

Definition at line 56 of file jvector.C.

Referenced by jdErr_Eval::run().

```
00057 {
00058   if(!L) return INF;
00059   double m = V[0];
00060   for(int i=1; i<L; ++i) { if(m < V[i]) m = V[i]; }
00061
```

4.41.2.18 double jVector::mean () const

Definition at line 100 of file jvector.C.

Referenced by Covariance().

```
00101   { return sum() / L; }
00102
```

4.41.2.19 double jVector::median () const

Definition at line 91 of file jvector.C.

```
00092   {
00093     // Find the middle element.  Have to sort first:
00094     jVector sorted(*this);
00095     JVqsort(sorted);
00096     int i = (int) floor(L/2);
00097     return ( (L%2) ? sorted.V[i] : (sorted.V[i] + sorted.V[i+1]) );
00098
```

4.41.2.20 double jVector::min () const

Definition at line 73 of file jvector.C.

```
00074 {
00075   if(!L) return INF;
00076   double m = V[0];
00077   for(int i=1; i<L; ++i) { if(m > V[i]) m = V[i]; }
00078
```

4.41.2.21 void jVector::mult (double *d*) [private]

4.41.2.22 jVector jVector::norm_unit () const [inline]

Returns a jVector of unit length;.

Definition at line 106 of file jvector.h.

Referenced by Envelope::CalcGradient().

```
00106                           { return *this/nrm(); }
00107
```

4.41.2.23 double jVector::nrm () const

Returns the norm (square of magnitude.

Definition at line 103 of file jvector.C.

Referenced by EnvelopeConstraint::AlignModeltoCOM(), ConstraintManager::TranslateModel(), gwNumerics::addVDWSegments(), mag(), and norm_unit().

```
00104   {
00105     double sumsqr = 0.0;
00106     for(int i=0; i<L; ++i) sumsqr += V[i]*V[i];
00107     return sumsqr;
00108
```

4.41.2.24 const jVector & jVector::operator *= (double *d*)

Definition at line 138 of file jvector.C.

```
00139   { for(int i=0; i<L; ++i) V[i] *= d; return *this; }
00140
```

4.41.2.25 jVector::operator double * () const

Definition at line 20 of file jvector.C.

```
00021 {
00022   double * d = 0;
00023   if(!V || L<1 || !(d = new double[L]))
00024     { cerr << "jVector::operator double *\n"; raise(SIGQUIT); }
00025
```

4.41.2.26 const jVector & jVector::operator%= (double *d*)

Definition at line 147 of file jvector.C.

```
00148   { for(int i=0; i<L; ++i) V[i] = fmod(V[i],d); return *this; }
00149
```

4.41.2.27 const jVector & jVector::operator%= (const jVector & *y*)

Definition at line 132 of file jvector.C.

```
00133   { return y; }
00134
```

4.41.2.28 const jVector & jVector::operator+= (const jVector & *y*)

Definition at line 119 of file jvector.C.

```
00120   { if(L == y.L) for(int i=0; i<L; ++i) V[i] += y.V[i]; return *this; }
00121
```

4.41.2.29 jVector jVector::operator- () [inline]

negation.

Definition at line 129 of file jvector.h.

```
00130   {
00131     jVector x(L);
00132     for(int i=0; i<L; ++i)
00133       x.V[i] = -V[i];
00134     return x;
00135
```

4.41.2.30 const jVector & jVector::operator-= (const jVector & *y*)

Definition at line 115 of file jvector.C.

```
00116   { if(L == y.L) for(int i=0; i<L; ++i) V[i] -= y.V[i]; return *this; }
00117
```

4.41.2.31 const jVector & jVector::operator/= (double *d*)

Definition at line 143 of file jvector.C.

```
00144   { if(d==0.0) return *this; for(int i=0; i<L; ++i) V[i] /= d; return *this; }
00145
```

4.41.2.32 const jVector & jVector::operator= (const double & *d*) [inline]

Definition at line 141 of file jvector.h.

```
00142   { return set(d); }
00143
```

4.41.2.33 const jVector & jVector::operator= (const jVector & *y*) [inline]

Definition at line 138 of file jvector.h.

```
00139   { Copy(y); return *this; }
00140
```

4.41.2.34 double & jVector::operator[] (int *i*) [inline]

Definition at line 121 of file jvector.h.

```
00122   {
00123     if(i<0||i>=L)
00124       { cerr << "jVector::operator[]\n"; raise(SIGQUIT); }
00125
```

4.41.2.35 const double & jVector::operator[] (int *i*) const [inline]

indexing.

Definition at line 115 of file jvector.h.

```
00116   {
00117     if(i<0||i>=L)
00118       { cerr << "jVector::operator[]\n"; raise(SIGQUIT); }
00119
```

4.41.2.36 const jVector & jVector::set (const double & *d*) [inline]

Definition at line 65 of file jvector.h.

```
00066   { for(int i=0; i<L; ++i) V[i]=d; return *this; }
00067
```

4.41.2.37 void jVector::set (int *i*, const double & *d*) [inline]

Definition at line 62 of file jvector.h.

Referenced by jVector(), operator=(), and zero().

```
00063   { V[i]=d; }
00064
```

4.41.2.38 double jVector::sum () const

Returns the sum of all the components.

Definition at line 29 of file jvector.C.

Referenced by mean(), jdRand_Eval::run(), and jdErr_Eval::run().

```
00030   {
00031     if(!L) return INF;
00032     double s=0.0;
00033     for(int i=0; i<L; ++i)
00034         s += V[i];
00035     return s;
00036
```

4.41.2.39 double jVector::sumsqr () const

Returns the sum of the squares.

Definition at line 38 of file jvector.C.

```
00039   {
00040     if(!L) return INF;
00041     double s=0.0;
00042     for(int i=0; i<L; ++i)
00043         s += V[i]*V[i];
00044     return s;
00045
```

4.41.2.40 const jVector & jVector::transpose () [inline]

Definition at line 82 of file jvector.h.

```
00082                               { vert = vert ^ 1; return *this; }
00083
```

4.41.2.41 jVector jVector::unit () const [inline]

Returns a jVector of unit length;.

Definition at line 112 of file jvector.h.

Referenced by EnvelopeConstraint::MoveModelbyGrad().

```
00112                         { return *this/mag(); }
00113
```

4.41.2.42 double jVector::variance () const

Returns the variance.

Definition at line 47 of file jvector.C.

```
00048   {
00049     if(!L) return INF;
00050     double s=0.0,ss=0.0;
00051     for(int i=0; i<L; ++i)
00052       { s += V[i]; ss += V[i]*V[i]; }
00053
```

4.41.2.43 const jVector & jVector::zero () [inline]

Definition at line 68 of file jvector.h.

Referenced by Envelope::CalcGradient(), Grid::Grid(), EnvelopeConstraint::MoveModelbyGrad(), Grid::ReSize(), and Envelope::Upsize().

```
00069   { return set(0.0); }
00070
```

4.41.3 Friends And Related Function Documentation

4.41.3.1 double Covariance (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 326 of file jvector.C.

```
00327   {
00328     if(x.L != y.L || x.L == 0)
00329         return INF;
00330
00331     double x_mean = x.mean();
00332     double y_mean = y.mean();
00333     double cov = 0.0;
00334
00335     for(int i=0; i<x.L; ++i)
00336         cov += (x[i] - x_mean)*(y[i] - y_mean);
00337     cov /= (x.L-1);
00338
00339     return cov;
00340
```

4.41.3.2 double JVAngle (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 152 of file jvector.C.

```
00153   { return acos( (x*y) / x.mag() / y.mag() ); }
00154
```

4.41.3.3 void JVqsort (jVector & *v*) [friend]

Definition at line 329 of file jvector.h.

Referenced by median().

```
00329                          { qsort((void *) v.V, v.L, sizeof(double), dcomp); }
00330
```

4.41.3.4 ostream& VBPrintCircle (ostream & *os*, const jVector & *x*, const double & *radius*, const DynString & *color*) [friend]

Definition at line 264 of file jvector.C.

```
00266   {
00267
00268     if((x.L>3) || (x.L<2)) return os;
00269     char old_delim = jVector::IODelimiter;
00270     jVector::IODelimiter = ',';
00271
00272     if(x.L==3)
00273       os << "circle (" << x << ") " << radius << " :" << color;
00274     else if(x.L==2)
00275       os << "circle (" << x << ",0) " << radius << " :" << color;
00276     os << '\n';
00277     jVector::IODelimiter = old_delim;
00278     return os;
00279
```

4.41.3.5 ostream& VBPrintLine (ostream & *os*, const jVector & *x*, const jVector & *y*, const DynString & *color*) [friend]

Definition at line 226 of file jvector.C.

```
00228   {
00229     if((x.L != y.L) || (x.L>3) || (y.L>3) || (x.L<1) || (y.L<1)) return os;
00230
00231     char old_delim = jVector::IODelimiter;
00232     jVector::IODelimiter = ',';
00233
00234     if(x.L==3)
00235       os << "line (" << x << ") (" << y << ") :" << color;
00236     else if(x.L==2)
00237       os << "line (" << x << ",0) (" << y << ",0) :" << color;
00238     else if(x.L==1)
00239       os << "line (" << x << "0,0) (" << y << "0,0) :" << color;
00240     os << '\n';
00241     jVector::IODelimiter = old_delim;
00242     return os;
00243
```

4.41.3.6 ostream& VBPrintPoint (ostream & *os*, const jVector & *x*, const DynString & *color*) [friend]

Definition at line 245 of file jvector.C.

```
00246   {
00247
00248     if((x.L>3) || (x.L<1)) return os;
00249     char old_delim = jVector::IODelimiter;
00250     jVector::IODelimiter = ',';
00251
00252     if(x.L==3)
```

```
00253       os << "point " << x << " :" << color;
00254     else if(x.L==2)
00255       os << "point " << x << ",0 :" << color;
00256     else if(x.L==1)
00257       os << "point " << x << ",0,0 :" << color;
00258     os << '\n';
00259     jVector::IODelimiter = old_delim;
00260     return os;
00261
```

4.41.3.7 int binsearch (const jVector & *v*, double *d*) [friend]

Definition at line 299 of file jvector.h.

```
00300 {
00301   // left index, right index, pivot, respectively.
00302   //
00303   int i=0;
00304   int j=v.L-1;
00305   int p;
00306
00307   while((i<=j) && (j>=0))
00308     {
00309       p = (j+i)/2;
00310       if(d < v.V[p]) j = p-1;
00311       else if(d > v.V[p]) i = p+1;
00312       else return p;
00313     }
00314
```

4.41.3.8 double euclid (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 155 of file jvector.C.

```
00156   {
00157     if(x.L != y.L) return -1.0;
00158     double sumproducts = 0.0;
00159     for(int i=0; i<x.L; ++i) sumproducts += pow(x.V[i]-y.V[i], 2.0);
00160     return sqrt(sumproducts);
00161
```

4.41.3.9 jVector operator * (double *d*, jVector *y*) [friend]

Definition at line 141 of file jvector.C.

```
00141                                       { return y*=d; }
00142
```

4.41.3.10 jVector operator * (jVector *y*, double *d*) [friend]

Definition at line 140 of file jvector.C.

```
00140                                       { return y*=d; }
00141
```

4.41.3.11 double operator ∗ (jVector *x*, const jVector & *y*) [friend]

Definition at line 124 of file jvector.C.

```
00125 {
00126   double product_sum=0.0;
00127   if(x.L == y.L) for(int i=0; i<x.L; ++i) product_sum += x.V[i]*y.V[i];
00128   return product_sum;
00129
```

4.41.3.12 int operator!= (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 152 of file jvector.h.

```
00153   { return !(x == y); }
00154
```

4.41.3.13 jVector operator% (jVector *x*, double *d*) [friend]

Definition at line 149 of file jvector.C.

```
00150   { return x%=d; }
00151
```

4.41.3.14 jVector operator% (jVector *x*, const jVector & *y*) [friend]

Definition at line 134 of file jvector.C.

```
00134                                                 { return x %= y; }
00135
```

4.41.3.15 jVector operator+ (jVector *x*, const jVector & *y*) [friend]

Definition at line 121 of file jvector.C.

```
00121                                                { return (x += y); }
00122
```

4.41.3.16 jVector operator- (jVector *x*, const jVector & *y*) [friend]

Definition at line 117 of file jvector.C.

```
00117                                                { return (x -= y); }
00118
```

4.41.3.17 jVector operator/ (jVector *y*, double *d*) [friend]

Definition at line 145 of file jvector.C.

```
00145                                        { return y/=d; }
00146
```

4.41.3.18 int operator< (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 154 of file jvector.h.

```
00155   {
00156     if(x.L < y.L) return 1;
00157     else if(x.L > y.L) return 0;
00158     // If the vectors are the same size:
00159     for(int i=x.L-1; i>=0; --i)
00160       {
00161         if(x.V[i] > y.V[i]) return 0;
00162         else if(x.V[i] < y.V[i]) return 1;
00163       }
00164
```

4.41.3.19 int operator<= (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 178 of file jvector.h.

```
00179   {
00180     if(x.L < y.L) return 1;
00181     else if(x.L > y.L) return 0;
00182     // If the vectors are the same size:
00183     for(int i=x.L-1; i>=0; --i)
00184       {
00185         if(x.V[i] > y.V[i]) return 0;
00186         else if(x.V[i] < y.V[i]) return 1;
00187       }
00188
```

4.41.3.20 int operator== (const jVector & *x*, const jVector & *y*) [friend]

Comparison operators.

Definition at line 145 of file jvector.h.

```
00146   {
00147     if((x.L != y.L) || (x.vert != y.vert)) return 0;
00148     for(int i=x.L-1; i>=0; --i)
00149       if(x.V[i] != y.V[i]) return 0;
00150     return 1;
00151
```

4.41.3.21 int operator> (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 166 of file jvector.h.

```
00167   {
00168     if(x.L > y.L) return 1;
00169     else if(x.L < y.L) return 0;
00170     // If the vectors are the same size:
00171     for(int i=x.L-1; i>=0; --i)
00172       {
00173         if(x.V[i] < y.V[i]) return 0;
00174         else if(x.V[i] > y.V[i]) return 1;
00175       }
00176
```

4.41.3.22 int operator>= (const jVector & *x*, const jVector & *y*) [friend]

Definition at line 190 of file jvector.h.

```
00191   {
00192     if(x.L > y.L) return 1;
00193     else if(x.L < y.L) return 0;
00194     // If the vectors are the same size:
00195     for(int i=x.L-1; i>=0; --i)
00196       {
00197         if(x.V[i] < y.V[i]) return 0;
00198         else if(x.V[i] > y.V[i]) return 1;
00199       }
00200
```

4.41.4 Member Data Documentation

4.41.4.1 int jVector::CreationLength = DEFAULTLEN [static]

Definition at line 15 of file jvector.C.

4.41.4.2 char jVector::IODelimiter = [static]

Definition at line 16 of file jvector.C.

4.41.4.3 int jVector::L [private]

Definition at line 291 of file jvector.h.

4.41.4.4 double * jVector::V [private]

Definition at line 290 of file jvector.h.

4.41.4.5 jVector jVector::Zero [static]

Definition at line 18 of file jvector.C.

4.41.4.6 usint jVector::vert [private]

Definition at line 293 of file jvector.h.

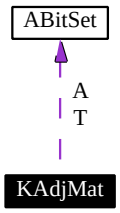
The documentation for this class was generated from the following files:

- **jvector.h**
- **jvector.C**

4.42 KAdjMat Class Reference

`#include <kadjmat.h>`

Collaboration diagram for KAdjMat:



Public Methods

- **KAdjMat** (int numvertices=**default_size**)
- **KAdjMat** (const KAdjMat &adj)
- **~KAdjMat** ()
- int **Copy** (const KAdjMat &adj)
- const KAdjMat& **operator=** (const KAdjMat &adj)
- int **Connect** (int i, int j)
- int **Disconnect** (int i, int j)
- const KAdjMat& **operator** *= (const KAdjMat &adj)
- const **ABitSet**& **operator**[] (int i) const
- const **ABitSet**& **operator()** (int i) const
- int **SetRow** (int i, const **ABitSet** &rowbits)
- int **SetCol** (int i, const **ABitSet** &colbits)

Static Public Methods

- int **DefaultSize** (int s)
- int **DefaultSize** ()

Static Public Attributes

- int **default_size**

Private Attributes

- int **N**
- **ABitSet*** **A**
- **ABitSet*** **T**

Friends

- KAdjMat **operator** * (KAdjMat x, const KAdjMat &y)
- KAdjMat **pow** (KAdjMat adj, int n)
- ostream& **operator<<** (ostream &os, const KAdjMat &adj)

4.42.1 Constructor & Destructor Documentation

4.42.1.1 KAdjMat::KAdjMat (int *numvertices* = default_size)

4.42.1.2 KAdjMat::KAdjMat (const KAdjMat & *adj*)

4.42.1.3 KAdjMat::~KAdjMat ()

4.42.2 Member Function Documentation

4.42.2.1 int KAdjMat::Connect (int *i*, int *j*) [inline]

Definition at line 38 of file kadjmat.h.
Referenced by sdGraph::AdjMatrix().

```
00039                   {
00040                     if(N<1 || i<0 || j<0 || i>=N || j>=N) return 0;
00041                     A[i].Set((uint) j);
00042                     T[j].Set((uint) i);
00043                     return 1;
00044
```

4.42.2.2 int KAdjMat::Copy (const KAdjMat & *adj*)

4.42.2.3 int KAdjMat::DefaultSize () [static]

Referenced by sdGraph::TravMatrices().

4.42.2.4 int KAdjMat::DefaultSize (int *s*) [static]

4.42.2.5 int KAdjMat::Disconnect (int *i*, int *j*) [inline]

Definition at line 47 of file kadjmat.h.

```
00048                   {
00049                     if(N<1 || i<0 || j<0 || i>=N || j>=N) return 0;
00050                     A[i].Clear((uint) j);
00051                     T[j].Clear((uint) i);
00052                     return 1;
00053
```

4.42.2.6 int KAdjMat::SetCol (int *i*, const ABitSet & *colbits*)

4.42.2.7 int KAdjMat::SetRow (int *i*, const ABitSet & *rowbits*)

Referenced by sdGraph::TravMatrices().

4.42.2.8 const KAdjMat& KAdjMat::operator *= (const KAdjMat & *adj*)

4.42.2.9 const ABitSet & KAdjMat::operator() (int *i*) const [inline]

Definition at line 67 of file kadjmat.h.

```
00067                                                   { return T[i]; }
00068
```

4.42.2.10 const KAdjMat& KAdjMat::operator= (const KAdjMat & *adj*)

4.42.2.11 const ABitSet & KAdjMat::operator[] (int *i*) const [inline]

Definition at line 65 of file kadjmat.h.

```
00065                                                   { return A[i]; }
00066
```

4.42.3 Friends And Related Function Documentation

4.42.3.1 KAdjMat operator * (KAdjMat *x*, const KAdjMat & *y*) [friend]

4.42.3.2 ostream& operator<< (ostream & *os*, const KAdjMat & *adj*) [friend]

4.42.3.3 KAdjMat pow (KAdjMat *adj*, int *n*) [friend]

4.42.4 Member Data Documentation

4.42.4.1 ABitSet * KAdjMat::A [private]

Definition at line 79 of file kadjmat.h.

4.42.4.2 int KAdjMat::N [private]

Definition at line 78 of file kadjmat.h.

4.42.4.3 ABitSet * KAdjMat::T [private]

Definition at line 80 of file kadjmat.h.

4.42.4.4 int KAdjMat::default_size [static]

Definition at line 24 of file kadjmat.h.

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

- **kadjmat.h**

4.43 Matrix Class Reference

`#include <matrix.h>`

Public Methods

- **Matrix** ()
- **Matrix** (int in_Rows, int in_Cols)
- **Matrix** (const Matrix &m)
- **~Matrix** ()
- Matrix& **operator=** (const Matrix &m)
- double& **operator()** (int row, int col)
- const double& **operator()** (int row, int col) const
- double& **operator()** (int row)
- const double& **operator()** (int row) const
- void **setSize** (int Rows, int Cols)
- int **getRows** () const
- int **getCols** () const
- void **zero** ()
- void **identity** ()
- void **swapRows** (int row1, int row2)
- void **swapCols** (int col1, int col2)
- Matrix **subMatrix** (int row1, int row2, int col1, int col2) const
- Matrix **subMatrix** (int row, int col) const

  excludes row and col only – the rest is transferred.

- Matrix **transpose** () const
- Matrix **t** () const
- Matrix **operator+** (const Matrix &m) const
- Matrix **operator-** (const Matrix &m) const
- Matrix **operator** * (double d) const
- Matrix **operator** * (const Matrix &m) const
- double **det** () const

  NOTE: this is an inefficient implementation (Gauss-Jordan would probably be faster).

- double **tr** () const
- double **Sum** () const
- double **SumAbsoluteValue** () const
- void **findEigenVectors** (Matrix &eigenValues, Matrix &eigenVectors) const
- ostream& **PrettyPrint** (ostream &os)

  pretty print normal numbers.

Private Attributes

- int **rows**
- int **cols**
- double* **data**

4.43.1 Constructor & Destructor Documentation

4.43.1.1 Matrix::Matrix ()

Definition at line 9 of file matrix.C.

```
00009               {
00010   rows = cols = 0;
00011   data = 0;
00012
```

4.43.1.2 Matrix::Matrix (int *in_Rows*, int *in_Cols*)

Definition at line 14 of file matrix.C.

```
00014                                       {
00015   rows = in_Rows;
00016   cols = in_Cols;
00017   data = new double[rows*cols];
00018
```

4.43.1.3 Matrix::Matrix (const Matrix & *m*)

Definition at line 20 of file matrix.C.

```
00020                               {
00021   rows = m.rows;
00022   cols = m.cols;
00023   data = new double[rows*cols];
00024   for (int i = 0; i < rows*cols; i++)
00025     data[i] = m.data[i];
00026
```

4.43.1.4 Matrix::~Matrix ()

Definition at line 28 of file matrix.C.

```
00028               {
00029   delete[] data;
00030   data = 0;
00031
```

4.43.2 Member Function Documentation

4.43.2.1 ostream & Matrix::PrettyPrint (ostream & *os*)

pretty print normal numbers.

Definition at line 388 of file matrix.C.

Referenced by EnvelopeConstraint::FindModelRotation().

```
00388                                         {
00389   for (int row = 1; row <= getRows(); row++) {
00390     os << setprecision(4) << "[ ";
```

```
00391     for (int col = 1; col <= getCols(); col++){
00392       if (fabs((*this)(row,col)> 1e-10))
00393         os << setw(8) << ($\ast$this)(row,col) << " ";
00394       else
00395         os << "0.000000 ";
00396     }
00397     os << "]" << endl;
00398   }
00399
```

4.43.2.2 double Matrix::Sum () const

Definition at line 159 of file matrix.C.

```
00159                         {
00160
00161   double rv = 0;
00162   for (int row = 1; row <= rows; row++)
00163     for (int col = 1; col <= cols; col++)
00164       rv += ($\ast$this)(row,col);
00165
00166   return rv;
00167
00168
```

4.43.2.3 double Matrix::SumAbsoluteValue () const

Definition at line 170 of file matrix.C.

```
00170                                     {
00171
00172   double rv = 0;
00173   for (int row = 1; row <= rows; row++)
00174     for (int col = 1; col <= cols; col++)
00175       rv += fabs((*this)(row,col));
00176
00177   return rv;
00178
00179
```

4.43.2.4 double Matrix::det () const

NOTE: this is an inefficient implementation (Gauss-Jordan would probably be faster).
Definition at line 332 of file matrix.C.

```
00332                         {
00333   if (rows != cols) {
00334     DIE("Matrix det() on non-square matrix");
00335   }
00336
00337   // Anton p.71
00338   if (rows == 1)
00339     return ($\ast$this)(1,1);
00340
00341   if (rows == 2)
00342     return ($\ast$this)(1,1)*($\ast$this)(2,2)-($\ast$this)(1,2)*($\ast$this)(2,1);
00343
```

```
00344   if (rows == 3) {
00345     return \
00346       ($\ast$this)(1,1)*($\ast$this)(2,2)*($\ast$this)(3,3) \
00347       + ($\ast$this)(1,2)*($\ast$this)(2,3)*($\ast$this)(3,1) \
00348       + ($\ast$this)(1,3)*($\ast$this)(2,1)*($\ast$this)(3,2) \
00349       - ($\ast$this)(1,3)*($\ast$this)(2,2)*($\ast$this)(3,1) \
00350       - ($\ast$this)(1,2)*($\ast$this)(2,1)*($\ast$this)(3,3) \
00351       - ($\ast$this)(1,1)*($\ast$this)(2,3)*($\ast$this)(3,2) ;
00352   }
00353
00354   DIE("Det is not working > 3x3 -- FIX");
00355   double sum = 0;
00356   for (int i = 1; i < cols; i++) {
00357
00358     if (odd(i))
00359       sum += ($\ast$this)(1,i) * subMatrix(1,i).det();
00360     else
00361       sum -= ($\ast$this)(1,i) * subMatrix(1,i).det();
00362
00363   }
00364
```

4.43.2.5 void Matrix::findEigenVectors (Matrix & *eigenValues*, Matrix & *eigenVectors*) const

Definition at line 230 of file matrix.C.

Referenced by Envelope::Calc_Eigenvectors(), and EnvelopeConstraint::FindModelRotation().

```
00230                                                                                 {
00231   if (rows != cols) {
00232     DIE("Matrix::findEigenVectors matrix must be square");
00233   }
00234   int i, j, row, col, n = rows, nrot = 0;
00235   Matrix a=*this,b(n,1),z(n,1);
00236   double tresh,theta,tau,t,s,h,g,c;
00237
00238   eigenValues.setSize(n,1);
00239   eigenVectors.setSize(n,n);
00240   eigenVectors.identity();
00241   // initialize b and d to diagonal
00242   for (i = 1; i <= n; i++) {
00243     b(i,1) = eigenValues(i,1) = a(i,i);
00244     z(i,1) = 0;
00245   }
00246   for (i = 1; i <= 50; i++) {
00247     double sum = 0;
00248     // sum off diagonal elements
00249     for (row = 1; row <= n-1; row++)
00250       for (col = row+1; col <= n; col++)
00251         sum += fabs(a(row,col));
00252     if (sum == 0) {  // normal return (relies on quadratic convergence to machine underflow)
00253       // sort eigenvalues and eigenvectors in descending order
00254       bool keepGoing=true;
00255       while (keepGoing) {
00256         keepGoing = false;
00257         for (row = 1; row <= n-1; row++)
00258           if (eigenValues(row,1) < eigenValues(row+1,1)) {
00259             eigenValues.swapRows(row,row+1);
00260             eigenVectors.swapCols(row,row+1);
00261             keepGoing = true;
00262           }
00263       }
```

```
00264       return;
00265     }
00266     if (i < 4)
00267       tresh = 0.2*sum/sq(n);  // first three steps
00268     else
00269       tresh = 0;              // thereafter
00270     for (row = 1; row <= n-1; row++)
00271       for (col = row+1; col <= n; col++) {
00272         g = 100*fabs(a(row,col));
00273         // after 4 sweeps, skip rotation if off-diagonal element is small
00274         if ((i > 4) && (fabs(eigenValues(row,1)+g) == fabs(eigenValues(row,1))) &&
00275                        (fabs(eigenValues(col,1)+g) == fabs(eigenValues(col,1))))
00276           a(row,col) = 0;
00277         else if (fabs(a(row,col)) > tresh) {
00278           h = eigenValues(col,1) - eigenValues(row,1);
00279           if (fabs(h)+g == fabs(h))
00280             t = a(row,col)/h;
00281           else {
00282             theta = 0.5*h/a(row,col);
00283             t = 1.0/(fabs(theta)+sqrt(1.0+sq(theta)));
00284             if (theta < 0.0)  t = -t;
00285           }
00286           c = 1.0/sqrt(1+sq(t));
00287           s = t*c;
00288           tau = s/(1.0+c);
00289           h = t*a(row,col);
00290           z(row,1) -= h;
00291           z(col,1) += h;
00292           eigenValues(row,1) -= h;
00293           eigenValues(col,1) += h;
00294           a(row,col) = 0;
00295           for (j = 1; j <= row-1; j++) {
00296             g = a(j,row);
00297             h = a(j,col);
00298             a(j,row) = g-s*(h+g*tau);
00299             a(j,col) = h+s*(g-h*tau);
00300           }
00301           for (j = row+1; j <= col-1; j++) {
00302             g = a(row,j);
00303             h = a(j,col);
00304             a(row,j) = g-s*(h+g*tau);
00305             a(j,col) = h+s*(g-h*tau);
00306           }
00307           for (j = col+1; j <= n; j++) {
00308             g = a(row,j);
00309             h = a(col,j);
00310             a(row,j) = g-s*(h+g*tau);
00311             a(col,j) = h+s*(g-h*tau);
00312           }
00313           for (j = 1; j <= n; j++) {
00314             g = eigenVectors(j,row);
00315             h = eigenVectors(j,col);
00316             eigenVectors(j,row) = g-s*(h+g*tau);
00317             eigenVectors(j,col) = h+s*(g-h*tau);
00318           }
00319           nrot++;
00320         }
00321       }
00322     for (int row = 1; row <= n; row++) {
00323       b(row,1) += z(row,1);
00324       eigenValues(row,1) = b(row,1);
00325       z(row,1) = 0;
00326     }
00327   }
00328
```

4.43.2.6 int Matrix::getCols () const [inline]

Definition at line 42 of file matrix.h.
Referenced by PrettyPrint(), operator ∗(), and operator<<().

```
00042                       { return cols; }
00043
```

4.43.2.7 int Matrix::getRows () const [inline]

Definition at line 41 of file matrix.h.
Referenced by PrettyPrint(), operator ∗(), and operator<<().

```
00041                       { return rows; }
00042
```

4.43.2.8 void Matrix::identity ()

Definition at line 104 of file matrix.C.
Referenced by EnvelopeConstraint::AlignPrincComp(), and findEigenVectors().

```
00104                       {
00105   if (rows != cols) {
00106     DIE("Matrix must be square to make identity");
00107   }
00108
```

4.43.2.9 Matrix Matrix::operator ∗ (const Matrix & *m*) const

Definition at line 212 of file matrix.C.

```
00212                                           {
00213   if (cols != m.rows) {
00214     DIE("Matrix multiplication left side cols != right side rows");
00215   }
00216   Matrix result(rows,m.cols);
00217   for (int row = 1; row <= rows; row++)
00218     for (int col = 1; col <= m.cols; col++) {
00219       double sum = 0;
00220       for (int i = 1; i <= cols; i++)
00221         sum += ($\ast$this)(row,i) * m(i,col);
00222       result(row,col) = sum;
00223     }
00224
```

4.43.2.10 Matrix Matrix::operator ∗ (double *d*) const

Definition at line 182 of file matrix.C.

```
00182                                     {
00183   Matrix result(rows,cols);
00184   for (int row = 1; row <= rows; row++)
```

```
00185     for (int col = 1; col <= cols; col++)
00186       result(row,col) = d * ($\ast$this)(row,col);
00187   return result;
00188
```

4.43.2.11 const double & Matrix::operator() (int *row*) const

Definition at line 75 of file matrix.C.

```
00075                                             {
00076   if (rows == 1 || cols == 1) {
00077     if ((row < 1) || (row > rows)) {
00078       DIE("Matrix access out of bounds");
00079     }
00080     return data[row-1];
00081
00082   } else {
00083     DIE("Matrix singular access on 2D Matrix");
00084     return data[row-1];
00085   }
00086
```

4.43.2.12 double & Matrix::operator() (int *row*)

Definition at line 60 of file matrix.C.

```
00060                                  {
00061   if (rows == 1 || cols == 1) {
00062     if ((row < 1) || (row > rows)) {
00063       DIE("Matrix access out of bounds");
00064     }
00065     return data[row-1];
00066
00067   } else {
00068     DIE("Matrix singular access on 2D Matrix");
00069     return data[row-1];
00070   }
00071
```

4.43.2.13 const double & Matrix::operator() (int *row*, int *col*) const

Definition at line 53 of file matrix.C.

```
00053                                                        {
00054   if ((row < 1) || (row > rows) || (col < 1) || (col > cols)) {
00055     DIE("Matrix access out of bounds");
00056   }
00057
```

4.43.2.14 double & Matrix::operator() (int *row*, int *col*)

Definition at line 46 of file matrix.C.

```
00046                                             {
00047   if ((row < 1) || (row > rows) || (col < 1) || (col > cols)) {
00048     DIE("Matrix access out of bounds");
00049   }
00050
```

4.43.2.15 Matrix Matrix::operator+ (const Matrix & *m*) const

Definition at line 190 of file matrix.C.

```
00190                                              {
00191   if ((cols != m.cols) || (rows != m.rows)) {
00192     DIE("Matrix addition sizes are different");
00193   }
00194
```

4.43.2.16 Matrix Matrix::operator- (const Matrix & *m*) const

Definition at line 201 of file matrix.C.

```
00201                                              {
00202   if ((cols != m.cols) || (rows != m.rows)) {
00203     DIE("Matrix subtraction sizes are different");
00204   }
00205
```

4.43.2.17 Matrix & Matrix::operator= (const Matrix & *m*)

Definition at line 33 of file matrix.C.

```
00033                                          {
00034   if (this == &m)  return *this;  // handle self-assignment
00035   if (data)
00036     delete[] data;
00037
00038   rows = m.rows;
00039   cols = m.cols;
00040   data = new double[rows*cols];
00041   for (int i = 0; i < rows*cols; i++)
00042     data[i] = m.data[i];
00043   return *this;
00044
```

4.43.2.18 void Matrix::setSize (int *Rows*, int *Cols*)

Definition at line 89 of file matrix.C.
Referenced by findEigenVectors().

```
00089                                       {
00090   if (data)
00091     delete[] data;
00092   rows = Rows;
00093   cols = Cols;
00094   data = new double[rows*cols];
00095
```

4.43.2.19 Matrix Matrix::subMatrix (int *row*, int *col*) const

excludes row and col only – the rest is transferred.
Definition at line 135 of file matrix.C.

```
00135                                                {
00136   Matrix result(rows-1,cols-1);
00137   int row2 = 1, col2 = 1;
00138   int row1 = 1, col1 = 1;
00139   for (row1 = 1; row1 <= rows; row1++) {
00140     col2 = 1;
00141     for (col1 = 1; col1 <= cols; col1++) {
00142       if ((row1 != row) && (col1 != col))
00143         result(row2,col2) = ($\ast$this)(row1,col1);
00144       if (col1 != col)  col2++;
00145     }
00146     if (row1 != row)  row2++;
00147   }
00148
```

4.43.2.20 Matrix Matrix::subMatrix (int *row1*, int *row2*, int *col1*, int *col2*) const

Referenced by det().

4.43.2.21 void Matrix::swapCols (int *col1*, int *col2*)

Definition at line 121 of file matrix.C.
Referenced by findEigenVectors().

```
00121                                        {
00122   for (int row = 1; row <= rows; row++)
00123     util_swap((*this)(row,col1),($\ast$this)(row,col2));
00124
```

4.43.2.22 void Matrix::swapRows (int *row1*, int *row2*)

Definition at line 116 of file matrix.C.
Referenced by findEigenVectors().

```
00116                                        {
00117   for (int col = 1; col <= cols; col++)
00118     util_swap((*this)(row1,col),($\ast$this)(row2,col));
00119
```

4.43.2.23 Matrix Matrix::t () const [inline]

Definition at line 81 of file matrix.h.
Referenced by EnvelopeConstraint::FindModelRotation().

```
00081                               {return transpose();}
00082
```

4.43.2.24 double Matrix::tr () const

Definition at line 368 of file matrix.C.
Referenced by EnvelopeConstraint::FindModelRotation().

```
00368                         {
00369
00370   if (rows == 1)
00371     return ($\ast$this)(1,1);
00372
00373   if (rows == 2)
00374     return ($\ast$this)(1,1)+($\ast$this)(2,2);
00375
00376   if (rows == 3) {
00377     return ($\ast$this)(1,1)+($\ast$this)(2,2)+($\ast$this)(3,3);
00378   }
00379
```

4.43.2.25 Matrix Matrix::transpose () const

Definition at line 151 of file matrix.C.
Referenced by t().

```
00151                               {
00152   Matrix result(cols,rows);
00153   for (int row = 1; row <= rows; row++)
00154     for (int col = 1; col <= cols; col++)
00155       result(col,row) = ($\ast$this)(row,col);
00156   return result;
00157
```

4.43.2.26 void Matrix::zero ()

Definition at line 97 of file matrix.C.
Referenced by Envelope::Calc_Eigenvectors(), and EnvelopeConstraint::FindModelRotation().

```
00097                   {
00098   for (int row = 1; row <= rows; row++)
00099     for (int col = 1; col <= cols; col++)
00100       //      (*this)(row,col) = 0;
00101       data[(col-1)+(row-1)*cols] = 0;
00102
```

4.43.3 Member Data Documentation

4.43.3.1 int Matrix::cols [private]

Definition at line 76 of file matrix.h.

4.43.3.2 double * Matrix::data [private]

Definition at line 77 of file matrix.h.

4.43.3.3 int Matrix::rows [private]

Definition at line 75 of file matrix.h.

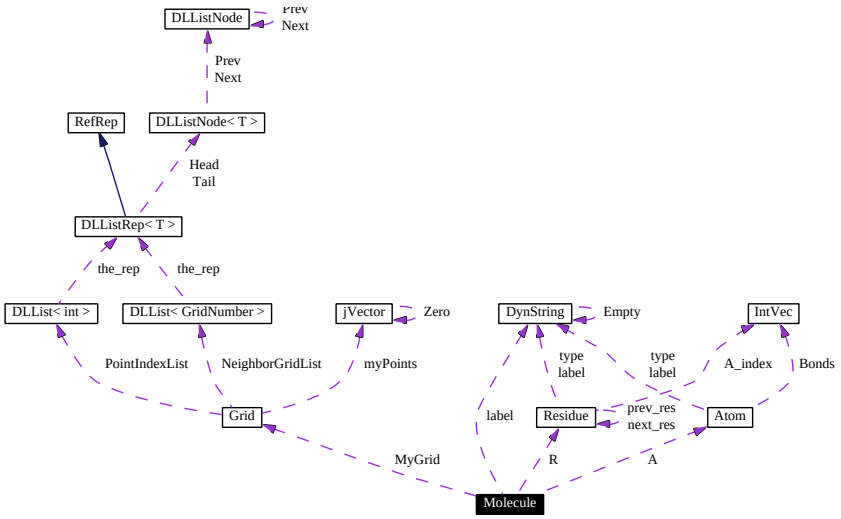
The documentation for this class was generated from the following files:

- matrix.h
- matrix.C

4.44 Molecule Class Reference

`#include <molecule.h>`

Collaboration diagram for Molecule:



Public Methods

- **Molecule** ()
- **Molecule** (int num_atoms)

  Constructs an atomic structure.

- **Molecule** (const **Atom** *atoms, int num_atoms)
- **Molecule** (const **jVector** *points_3d, int num_pos)
- **Molecule** (const Molecule &mol)
- **~Molecule** ()
- int **Number** () const

  Reports the number of atoms in the molecule.

- int **NumRes** () const

  Reports the number of residues in the molecule.

- const Molecule& **operator=** (const Molecule &mol)

  Assignment (calls the **Copy***()* (p. 361) *function).*

- const **Atom**& **getAtom** (int i) const

  access to an atom.

- void **setAtom** (int i, const **Atom** &a)

  make atom a the i'th atom.

- const **Grid**& **getGrid** () const

  talk to the grid (note const).

- const **Atom**& **operator**[ ] (int i) const

  Access an atom in the molecule by the atom id. Note, the id and index of an atom are the same.

- **Atom**& **operator**[ ] (int i)

  Set an atom in the molecule by the atom id. Note, the id and index of an atom are the same.

- **DLList**<int> **approx_neighbor_i** (const **jVector** &p, double distance) const

  grid interface to neighbor atoms (Approx, fast).

- **DLList**<int> **neighbor_i** (const **jVector** &p, double distance) const

  grid interface to neighbor atoms (Exact, same as above, but checked).

- const **jVector**& **operator()** (int i) const

  returns position of atom i.

- const **jVector**& **getPosition** (int i) const

  returns position of atom i.

- void **setPosition** (int i, const **jVector** &newPos)

  sets position of atom i.

- **jVector getCOM** () const

  return center of mass.

- double **VDW** (int i) const

  VDW for atom i'.

- const **IntVec**& **GetBonds** (int atom) const

  returns list of bonds in an atom.

- double **MSD** (int a1, int a2) const

  return physical Minimum Separation Distance between two atoms.

- int **Output** (ostream &os) const
- int **Input** (istream &is)
- int **OutputBinary** (ostream &os) const
- int **InputBinary** (istream &is)
- istream& **DynRead** (istream &is)
- int **setRandom** (void)

  Generates random positions for Num' atoms.

- int **readXmol** (istream &is)

  Reads in a molecule from an xyz formatted file. Returns the number of atoms read in.

- int **readFasta** (istream &is)

  Reads in a molecule from an xyz formatted file. Returns the number of atoms read in.

- void **writeXmol** (ostream &os) const

Writes a file in xmol xyz format. This version puts the atom int atom number as the first Xmol field.

- void **writeXmolData** (ostream &os, int *data, **DynString** comment) const

  Writes a client specified integer for each atom in the output Xmol formatted file.

- void **writeXmolData** (ostream &os, double *data, **DynString** comment) const

  Writes a client specified double for each atom in the output Xmol formatted file.

- void **Bounds** (double &minx, double &miny, double &minz,\double &maxx, double &maxy, double &maxz) const

  writes the upper and lower bounds in 3 dimensions by reference.

- **MAKECOMPARISONS_NO_COMPARE** (Molecule)
- **MAKE_IO_STREAM_OPERATORS** (Molecule)
- **MAKE_BIN_STREAM_OPERATORS** (Molecule)

Static Public Methods

- void **skipXmol** (istream &is)

  skip a frame in a xmol format file stream.

Static Public Attributes

- double **grid_delta** = 3.7

  expected inter-atomic width.

- double **grid_max_radius** = 250

  expected max molecule size.

Protected Methods

- void **Allocate** (int n)

  Allocates space for n atoms.

- void **Deallocate** ()

  Deallocates the space.

- void **Reallocate** (int n)

  Destroys the currently allocated space if n!=N and creates space for n atoms.

- void **Copy** (const Molecule &mol)

  Copies a molecule...

Protected Attributes

- **DynString label**

  label for the molecule.

- int **N**

  The number of atoms in the structure.

- int **P**

  The number of residues in the structure.

- **Atom∗ A**

  A is a vector of atoms in the molecule (0-> N-1).

- **Residue∗ R**

  R is a vector of residues in the molecule (0-> P-1).

- **Grid∗ MyGrid**

  My own grid to keep track of atom positions.

Friends

- class **Residue**

4.44.1 Detailed Description

//

File: Molecule.h

This class manages molecules. A separate class called atom is used to represent the data for particular atoms.

//

Definition at line 32 of file molecule.h.

4.44.2 Constructor & Destructor Documentation

4.44.2.1 Molecule::Molecule ()

Definition at line 3 of file molecule.C.

```
00004   { N = 0; A = 0; R = 0;MyGrid =0; }
00005
```

4.44.2.2 Molecule::Molecule (int *num_atoms*)

Constructs an atomic structure.

4.44.2.3 Molecule::Molecule (const Atom * *atoms*, int *num_pos*)

Definition at line 18 of file molecule.C.

```
00019   {
00020     N=0; A=0;
00021     Allocate(num_pos);
00022     for(int i=0; i<N; ++i)
00023         A[i] = atoms[i];
00024     R = 0;
00025
```

4.44.2.4 Molecule::Molecule (const jVector * *points_3d*, int *num_pos*)

Definition at line 6 of file molecule.C.

```
00006                                                    {
00007   N=0; A=0;
00008   Allocate(num_pos);
00009   for(int i=0; i<N; ++i)
00010     {
00011       //     A[i].position() = points_3d[i];
00012       MyGrid->setPosition(i, points_3d[i]);
00013       A[i].ID() = i;
00014     }
00015
```

4.44.2.5 Molecule::Molecule (const Molecule & *mol*)

Definition at line 27 of file molecule.C.

```
00028   {
00029     N=0; A=0;
00030     Copy(mol);
00031
```

4.44.2.6 Molecule::~Molecule ()

Definition at line 33 of file molecule.C.

```
00034   { Deallocate(); }
00035
```

4.44.3 Member Function Documentation

4.44.3.1 void Molecule::Allocate (int *n*) [protected]

Allocates space for n atoms.

Definition at line 36 of file molecule.C.

Referenced by Molecule(), and Reallocate().

```
00036                           {
00037   N = (n < 1) ? 0 : n;
```

```
00038   A = 0;
00039   if(N)
00040     {
00041       if(!(A = new Atom[N]))
00042         N = 0;
00043       else
00044         for(int i=0; i<N; ++i)
00045           A[i].ID() = i;
00046     }
00047
```

4.44.3.2 void Molecule::Bounds (double & *minx*, double & *miny*, double & *minz*, \double & *maxx*, double & *maxy*, double & *maxz*) const

writes the upper and lower bounds in 3 dimensions by reference.

Definition at line 92 of file molecule.C.

Referenced by Envelope::Build().

```
00093                                                                    {
00094
00095   //  minx = miny  = minz = INF;
00096   //  maxx = maxy  = maxz = (-1) * INF;
00097   jVector temp;
00098
00099   temp = MyGrid->getPosition(0);
00100   minx = maxx = temp[0];
00101   miny = maxy = temp[1];
00102   minz = maxz = temp[2];
00103
00104
00105   for (int i = 1; i < N ; i++){
00106     // assumes 2 atoms
00107     temp = MyGrid->getPosition(i);
00108
00109     if (temp[0] < minx)
00110       minx = temp[0];
00111     else
00112       if (temp[0] > maxx)
00113         maxx = temp[0];
00114
00115     if (temp[1] < miny)
00116       miny = temp[1];
00117     else
00118       if (temp[1] > maxy)
00119         maxy = temp[1];
00120
00121     if (temp[2] < minz)
00122       minz = temp[2];
00123     else
00124       if (temp[2] > maxz)
00125         maxz = temp[2];
00126
00127   }
00128
```

4.44.3.3 void Molecule::Copy (const Molecule & *mol*) [protected]

Copies a molecule...

Definition at line 77 of file molecule.C.

Referenced by Molecule(), and operator=().

```
00077                                       {
00078   Reallocate(mol.N);
00079   for(int i=0; i<N; ++i){
00080     A[i] = mol.A[i];
00081     MyGrid->setPosition(i, mol.MyGrid->getPosition(i));
00082   }
00083   if (mol.R){ // if mol is using residues
00084     P = mol.P;
00085     R = new Residue[P];
00086     for(int i=0; i<P; ++i){
00087       R[i] = mol.R[i];
00088     }
00089   }
00090
```

4.44.3.4 void Molecule::Deallocate () [protected]

Deallocates the space.

Definition at line 51 of file molecule.C.

Referenced by Reallocate(), and ~Molecule().

```
00051                           {
00052   if (A) delete[] A;
00053   if (R) delete[] R;
00054
00055
00056   if (MyGrid) delete MyGrid;
00057   N = 0;
00058   A = 0;
00059   R = 0;
00060
```

4.44.3.5 istream & Molecule::DynRead (istream & *is*)

Definition at line 259 of file molecule.C.

```
00260   {
00261     DLList<jVector> L;
00262     jVector input_vec(3);
00263
00264     while(input_vec.dynread(is))
00265         L.AddToBack(input_vec);
00266
00267     Reallocate(L.Number());
00268
00269     for(int i=0; L; ++i)
00270         MyGrid->setPosition(i, L.RemoveFromFront());
00271
00272     return is;
00273
```

4.44.3.6 const IntVec & Molecule::GetBonds (int *atom*) const [inline]

returns list of bonds in an atom.

Definition at line 235 of file molecule.h.

```
00236 { return A[atom].Bonds;}
00237
```

4.44.3.7 int Molecule::Input (istream & *is*)

Definition at line 218 of file molecule.C.

```
00219   {
00220     int n=0;
00221     is >> n;
00222
00223     if(n<1)
00224       return 0;
00225
00226     Reallocate(n);
00227
00228     jVector tempPos;
00229     for(int i=0; i<n; ++i){
00230       is >> tempPos ;
00231       MyGrid->setPosition(i, tempPos);
00232     }
00233
```

4.44.3.8 int Molecule::InputBinary (istream & *is*)

Definition at line 246 of file molecule.C.

```
00247   {
00248     // Read in the number of atoms:
00249     is >= N;
00250     Reallocate(N);
00251     jVector tempPos;
00252     for(int i=0; i<N; ++i){
00253       is >= tempPos;
00254       MyGrid->setPosition(i,tempPos);
00255     }
00256
```

4.44.3.9 Molecule::MAKECOMPARISONS_NO_COMPARE (Molecule)

4.44.3.10 Molecule::MAKE_BIN_STREAM_OPERATORS (Molecule)

4.44.3.11 Molecule::MAKE_IO_STREAM_OPERATORS (Molecule)

4.44.3.12 double Molecule::MSD (int *a1*, int *a2*) const

return physical Minimum Separation Distance between two atoms.
Definition at line 154 of file molecule.C.

```
00154                                       {
00155   double d = 1;
00156
00157
```

```
00158   //  cerr << "msd: a1 " << a1 << "msd: a2 " << a2 << endl;
00159 //   if (A[a1].Bonded(a2)){
00160 //     d=  A[a1].cov_dist()+A[a2].cov_dist();
00161 //     cerr << "BONDED: a1=" ;
00162 //   } else {
00163 //        d = A[a1].VDW()+A[a2].VDW();
00164 //     //    cerr << "NONBONDED: a1=" ;
00165 //   }
00166 // //   cerr << A[a1].Label() << ", a2=" << A[a2].Label() << ", msd=" << d << endl;
00167 // //   cerr << "atom " << a1 <<  " bonds: " << GetBonds(a1) << endl;
00168 // //   cerr << "atom " << a2 <<  " bonds: " << GetBonds(a2) << endl;
00169 // //   cerr << endl;
00170
00171
00172   int r1 = A[a1].residue_number;
00173   int r2 = A[a2].residue_number;
00174
00175
00176   if ((R[r1].N_chain_atom == a1) &&
00177       (R[r2].N_chain_atom == a2))
00178     {
00179             d=  2.6;
00180             //d= 2.7;
00181     }
00182   else if ((R[r1].CA_chain_atom == a1) &&
00183            (R[r2].CA_chain_atom == a2))
00184     {
00185             d = 3.6;
00186             //d= 3.7;
00187     }
00188   else if ((R[r1].O_chain_atom == a1) &&
00189            (R[r2].O_chain_atom == a2))
00190     {
00191       d = 2.8;
00192     }
00193   else if ((R[r1].C_chain_atom == a1) &&
00194            (R[r2].C_chain_atom == a2))
00195     {
00196             d = 2.8;
00197             //d= 2.9;
00198     }
00199   else if ((R[r1].CB_chain_atom == a1) &&
00200            (R[r2].CB_chain_atom == a2))
00201     {
00202       d = 3.5;
00203     }
00204
```

4.44.3.13 int Molecule::NumRes () const [inline]

Reports the number of residues in the molecule.
Definition at line 199 of file molecule.h.

```
00200 { return P; }
00201
```

4.44.3.14 int Molecule::Number () const [inline]

Reports the number of atoms in the molecule.

Definition at line 196 of file molecule.h.

Referenced by Envelope::Build(), ConstraintManager::ConstraintManager(), ConstraintManager::RandomPerturb(), ConstraintManager::TurnAtomOn(), ConstraintManager::TurnAtomsOff(), Envelope::WriteDensity(), ConstraintManager::clearConstraints(), ConstraintManager::getAssocAtoms(), ConstraintManager::getStressList(), ConstraintManager::numAtoms(), ConstraintManager::resetMoved(), ConstraintManager::setMolecule(), ConstraintManager::setPosition(), and ConstraintManager::stress().

```
00197 { return N; }
00198
```

4.44.3.15 int Molecule::Output (ostream & *os*) const

Definition at line 211 of file molecule.C.

```
00212   {
00213     os << N << '\n';
00214     for(int i=0; i<N; ++i)
00215         os << MyGrid->getPosition(i) << '\n';
00216     return N;
00217
```

4.44.3.16 int Molecule::OutputBinary (ostream & *os*) const

Definition at line 236 of file molecule.C.

```
00236                                           {
00237     // Write the number of atoms in the molecule:
00238     os <= N;
00239
00240     // Write out all the positions:
00241     for(int i=0; i<N; ++i)
00242         os <= MyGrid->getPosition(i);
00243     return N;
00244
```

4.44.3.17 void Molecule::Reallocate (int *n*) [protected]

Destroys the currently allocated space if n!=N and creates space for n atoms.

Definition at line 62 of file molecule.C.

Referenced by Copy(), DynRead(), Input(), InputBinary(), readFasta(), and readXmol().

```
00062                                   {
00063   if(N == n)
00064     return;
00065   if (R) {
00066     WARN ("ACK! LOSING RESIDUE INFORMATION");
00067     cerr << "Reallocating from " << N << " atoms to " << n << " atoms." << endl;
00068   }
00069
```

4.44.3.18 double Molecule::VDW (int *i*) const [inline]

VDW for atom i‘.

Definition at line 223 of file molecule.h.

Referenced by ConstraintManager::getRadius().

```
00224 { return A[i].VDW();}
00225
```

4.44.3.19 DLList< int > Molecule::approx_neighbor_i (const jVector & *p*, double *distance*) const [inline]

grid interface to neighbor atoms (Approx, fast).

Definition at line 229 of file molecule.h.

Referenced by ConstraintManager::approx_neighbor_i().

```
00230 { return MyGrid->approx_neighbor_i(p,distance); }
00231
```

4.44.3.20 const Atom & Molecule::getAtom (int *i*) const [inline]

access to an atom.

Definition at line 202 of file molecule.h.

Referenced by Envelope::Build().

```
00203 { return A[i]; }
00204
```

4.44.3.21 jVector Molecule::getCOM () const

return center of mass.

Definition at line 141 of file molecule.C.

```
00142 {
00143   jVector rv(3,0);
00144
00145   for(int i=0; i<N; ++i){
00146     rv += MyGrid->getPosition(i);
00147   }
00148
```

4.44.3.22 const Grid & Molecule::getGrid () const [inline]

talk to the grid (note const).

Definition at line 205 of file molecule.h.

```
00206 { return *MyGrid; }
00207
```

4.44.3.23 const jVector & Molecule::getPosition (int *i*) const [inline]

returns position of atom i.

Definition at line 220 of file molecule.h.

Referenced by Output(), and OutputBinary().

```
00221 { return MyGrid->getPosition(i); }
00222
```

4.44.3.24 DLList< int > Molecule::neighbor_i (const jVector & *p*, double *distance*) const [inline]

grid interface to neighbor atoms (Exact, same as above, but checked).

Definition at line 232 of file molecule.h.

Referenced by ConstraintManager::neighbor_i().

```
00233 { return MyGrid->neighbor_i(p,distance); }
00234
```

4.44.3.25 const jVector & Molecule::operator() (int *i*) const [inline]

returns position of atom i.

Definition at line 217 of file molecule.h.

```
00218 { return MyGrid->getPosition(i); }
00219
```

4.44.3.26 const Molecule & Molecule::operator= (const Molecule & *mol*)

Assignment (calls the **Copy**() (p. 361) function).

Definition at line 138 of file molecule.C.

```
00139   { Copy(mol); return *this; }
00140
```

4.44.3.27 Atom & Molecule::operator[ ] (int *i*) [inline]

Set an atom in the molecule by the atom id. Note, the id and index of an atom are the same.

Definition at line 214 of file molecule.h.

```
00215 { return A[i]; }
00216
```

4.44.3.28 const Atom & Molecule::operator[ ] (int *i*) const [inline]

Access an atom in the molecule by the atom id. Note, the id and index of an atom are the same.

Definition at line 211 of file molecule.h.

```
00212 { return A[i]; }
00213
```

4.44.3.29 int Molecule::readFasta (istream & *is*)

Reads in a molecule from an xyz formatted file. Returns the number of atoms read in.

Definition at line 302 of file molecule.C.

Referenced by main().

```
00303 {
00304   int res_count = 0;
00305   int tot_residues = 0;
00306   int atom_count = 0;
00307   int res_type_temp = UNINITIALIZED_RES_ID;
00308   DLList<DynString *> ResList;
00309   DLList<DynString *>::ConstIterator it;
00310   DynString * DynPtr;
00311
00312   DynString str,previous; // strings
00313   // ASSUMES CODE_1 SERIES
00314   previous.ReadLine(is); // read comment line
00315   label = previous;
00316
00317
00318   while(is.good()){
00319     str.Clear();
00320     str.GetChar(is);
00321     str.ToUpper();
00322     res_type_temp = ResidueRegistry::CheckType_code_1(str);
00323     if (res_type_temp > -1) {// its a real residue code
00324       res_count++;
00325       atom_count += ResidueRegistry::numAtoms[res_type_temp];
00326       DynPtr = new DynString(str);
00327       ResList.Add(DynPtr);
00328     }
00329   }
00330
00331   // set up molecule
00332   Reallocate(atom_count);
00333   P = res_count;
00334   R = new Residue[P];
00335
00336   tot_residues = res_count;
00337   res_count = 0;
00338   atom_count = 0;
00339
00340   // iterate over list, create molecule
00341   for(it = ResList.GetConstIterator(); it; ++it){
00342     DynPtr = ($\ast$it);
00343     R[res_count].ID() = res_count;
00344     ResidueRegistry::SetType_code_1(R[res_count], *DynPtr);
00345     if (tot_residues == 1)
00346       R[res_count].Setup(*this, atom_count,0,0); // this configures atoms
00347     else if (res_count == 0)
00348       atom_count = R[res_count].Setup(*this, atom_count,0,&(R[1]));
00349     else if (res_count == tot_residues-1)
00350       atom_count = R[res_count].Setup(*this, atom_count, \
00351                                       &(R[tot_residues-2]),0);
00352     else
00353       // this configures atoms
00354       atom_count = R[res_count].Setup \
```

```
00355         (*this, atom_count,&(R[res_count-1]),&(R[res_count+1]));
00356
00357     res_count++;
00358   }
00359
00360   // delete the DynStrings
00361   while (ResList.Number()){
00362     DynPtr = ResList.Remove();
00363     delete DynPtr;
00364   }
00365
```

4.44.3.30 int Molecule::readXmol (istream & *is*)

Reads in a molecule from an xyz formatted file. Returns the number of atoms read in.

Definition at line 373 of file molecule.C.

Referenced by main(), and ConstraintManager::readNewMoleculeXmol().

```
00374 {
00375
00376   DynString str,previous; // strings
00377   jVector tempPos(3); // 3 element vector
00378   char cJunk[256];
00379
00380
00381   previous.ReadLine(is); // read num atoms
00382   Reallocate(previous.ToInt()); // make me a molecule with this many atoms available
00383
00384   str.ReadLine(is); // read comment
00385   label = str;
00386
00387     //    cerr << "Read: " << label << endl;
00388
00389     // Read in the atoms (the number of atoms is specified on
00390     // the line before the -- line:
00391     //
00392
00393   for(int i=0; i<N; i++)
00394     {
00395       if(!is.good())
00396         {
00397           cerr << "Warning: read in " << i << ", not "
00398                << N << " atoms.\n";
00399           DIE("this is a fatal error; check file");
00400         }
00401
00402       //        is >> A[i].Type() >> tempPos ;
00403       // label removed from front -- now at the end
00404       is >> str >> tempPos ;
00405       is.getline(cJunk,256); //scrap rest -o- line!
00406       MyGrid->setPosition(i, tempPos);
00407       //      cerr << "setting type on atom " << i << " to " << str << endl;
00408       AtomRegistry::SetType(A[i],str, str);
00409     }
00410
```

4.44.3.31 void Molecule::setAtom (int *i*, const Atom & *a*) [inline]

make atom a the i'th atom.

Definition at line 208 of file molecule.h.

```
00209 { A[i] = a; }
00210
```

4.44.3.32 void Molecule::setPosition (int *i*, const jVector & *newPos*) [inline]

sets position of atom i.

Definition at line 226 of file molecule.h.
Referenced by ConstraintManager::setPosition().

```
00227 { MyGrid->setPosition(i, newPos); }
00228
```

4.44.3.33 int Molecule::setRandom (void)

Generates random positions for Num' atoms.

Definition at line 275 of file molecule.C.
Referenced by main().

```
00275                       {
00276   // Generates random positions for Num' atoms
00277   //  RanInt randomPosition((-1)*Num/3, Num/3);
00278
00279   RanInt randomPosition(-50, 50);
00280   // actually want double, but int is reproducible across systems
00281   jVector tempPos(3); // 3 element vector
00282
00283   ofstream out("random_start.xmol");
00284   out << N << endl;
00285   out << "Random Start: "<< label << endl;
00286
00287   for(int i=0; i<N; i++)
00288     {
00289       tempPos[0] = randomPosition.Get();
00290       tempPos[1] = randomPosition.Get();
00291       tempPos[2] = randomPosition.Get();
00292
00293       MyGrid->setPosition(i, tempPos);
00294
00295       out << A[i].type << tempPos << "\t  9" << endl;
00296     }
00297
```

4.44.3.34 void Molecule::skipXmol (istream & *is*) [static]

skip a frame in a xmol format file stream.

Definition at line 444 of file molecule.C.
Referenced by jdErr_Eval::run().

```
00445 {
00446
```

```
00447     DynString previous; // strings
00448     char cJunk[256];
00449
00450
00451     previous.ReadLine(is); // read num atoms
00452     int lines = previous.ToInt();
00453     is.getline(cJunk,256); // pull \n
00454
00455     is.getline(cJunk,256); //scrap comment
00456
00457     for(int i=0; i<lines; i++)
00458       {
00459         if(!is.good())
00460           {
00461             cerr << "Warning: skipXmol: read in " << i << ", not "
00462                  << lines << " atoms.\n";
00463             DIE("this is a fatal error; check file");
00464           }
00465
00466         is.getline(cJunk,256); //scrap each line
00467
00468       }
00469
```

4.44.3.35 void Molecule::writeXmol (ostream & *os*) const

Writes a file in xmol xyz format. This version puts the atom int atom number as the first Xmol field.

Definition at line 413 of file molecule.C.

Referenced by main().

```
00413                                          {
00414   os << N << '\n';
00415   os << label << "\n";
00416   for(int i=0; i<N; ++i)
00417     os  << A[i].type << " " << MyGrid->getPosition(i) << " 9"  << endl;
00418
```

4.44.3.36 void Molecule::writeXmolData (ostream & *os*, double * *data*, DynString *new_comment*) const

Writes a client specified double for each atom in the output Xmol formatted file.

Definition at line 432 of file molecule.C.

```
00433 {
00434   // Warning. using wrong data array can cause seg fault, cf above.
00435
00436   os << N << '\n';
00437   os << new_comment << "\n";
00438   for(int i=0; i<N; ++i)
00439     os << A[i].type  <<" "<< MyGrid->getPosition(i) << ' ' << data[i] << '\n';
00440
00441
```

4.44.3.37 void Molecule::writeXmolData (ostream & *os*, int * *data*, DynString *new_comment*) const

Writes a client specified integer for each atom in the output Xmol formatted file.

Definition at line 420 of file molecule.C.

```
00420                                                                                {
00421   // Warning.  This assumes that data points to at least N integers
00422   // I'm not sure a good way to make this safer. -jd
00423
00424   os << N << '\n';
00425   os << new_comment << "\n";
00426   for(int i=0; i<N; ++i)
00427     os << A[i].type <<" "<< MyGrid->getPosition(i) << ' ' << data[i] << endl;
00428
00429
```

4.44.4 Friends And Related Function Documentation

4.44.4.1 class Residue [friend]

Definition at line 35 of file molecule.h.

4.44.5 Member Data Documentation

4.44.5.1 Atom * Molecule::A [protected]

A is a vector of atoms in the molecule (0-> N-1).

Definition at line 170 of file molecule.h.

4.44.5.2 Grid * Molecule::MyGrid [protected]

My own grid to keep track of atom positions.

Definition at line 176 of file molecule.h.

4.44.5.3 int Molecule::N [protected]

The number of atoms in the structure.

Definition at line 164 of file molecule.h.

4.44.5.4 int Molecule::P [protected]

The number of residues in the structure.

Definition at line 167 of file molecule.h.

4.44.5.5 Residue * Molecule::R [protected]

R is a vector of residues in the molecule (0-> P-1).

Definition at line 173 of file molecule.h.

4.44.5.6 double Molecule::grid_delta = 3.7 [static]

expected inter-atomic width.

Definition at line 78 of file globals.C.

4.44.5.7 double Molecule::grid_max_radius = 250 [static]

expected max molecule size.

Definition at line 79 of file globals.C.

4.44.5.8 DynString Molecule::label [protected]

label for the molecule.

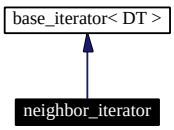
Definition at line 161 of file molecule.h.

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

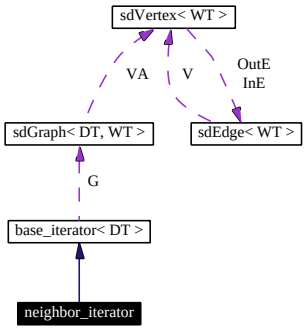
- **molecule.h**
- **globals.C**
- **molecule.C**

4.45 neighbor_iterator Class Reference

Inheritance diagram for neighbor_iterator



Collaboration diagram for neighbor_iterator:



Public Methods

- **neighbor_iterator** (int reverse=0)
- **neighbor_iterator** (const neighbor_iterator &ni)
- **~neighbor_iterator** ()
- void **reset** ()
- **operator int** () const
- const **base_iterator**& **operator++** ()
- int **operator** * () const

Protected Methods

- **neighbor_iterator** (**sdGraph**< DT, WT > *g, int vndx, int reverse=0)

Friends

- class **sdGraph**< **DT**, **WT** >

4.45.1 Constructor & Destructor Documentation

4.45.1.1 neighbor_iterator::neighbor_iterator (sdGraph< DT, WT > * *g*, int *vndx*, int *reverse* = 0) [inline, protected]

Definition at line 58 of file sdgiter.inc.

```
00059                   : base_iterator(g,vndx,reverse) { reset(); }
00060
```

4.45.1.2 neighbor_iterator::neighbor_iterator (int *reverse* = 0) [inline]

Definition at line 61 of file sdgiter.inc.

```
00062                                : base_iterator<DT>(reverse) { reset(); }
00063
```

4.45.1.3 neighbor_iterator::neighbor_iterator (const neighbor_iterator & *ni*) [inline]

Definition at line 65 of file sdgiter.inc.

```
00066                                : base_iterator<DT>(ni) {}
00067
```

4.45.1.4 neighbor_iterator::~neighbor_iterator () [inline]

Definition at line 70 of file sdgiter.inc.

```
00070                                   { /* empty */ }
00071
```

4.45.2 Member Function Documentation

4.45.2.1 int neighbor_iterator::operator * () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 95 of file sdgiter.inc.

```
00096                {
00097                  if((vid==-1) || (vstart==-1)) return -1;
00098                  if(!rev) { if(vid>=G->VA[vstart]->NumO) return -1; }
00099
```

4.45.2.2 neighbor_iterator::operator int () const [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 82 of file sdgiter.inc.

```
00083                {
00084                  if(vid == -1) return 0;
00085                  if(!rev) return (vid < G->VA[vstart]->NumO);
00086                  return (vid < G->VA[vstart]->NumI);
00087
```

4.45.2.3 const base_iterator & neighbor_iterator::operator++ () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).
Definition at line 89 of file sdgiter.inc.

```
00090                {
00091                  if(vid != -1) ++vid;
00092                  return *this;
00093
```

4.45.2.4 void neighbor_iterator::reset () [inline, virtual]

Reimplemented from **base_iterator** (p. 65).

Definition at line 72 of file sdgiter.inc.

```
00073                  {
00074                    if(bad()) vid = -1;
00075                    else
00076                      {
00077                        if(!rev) vid = (G->VA[vstart]->NumO > 0) ? 0 : -1;
00078                        else vid = (G->VA[vstart]->NumI > 0) ? 0 : -1;
00079                      }
00080
```

4.45.3 Friends And Related Function Documentation

4.45.3.1 class sdGraph [friend]

Reimplemented from **base_iterator** (p. 66).

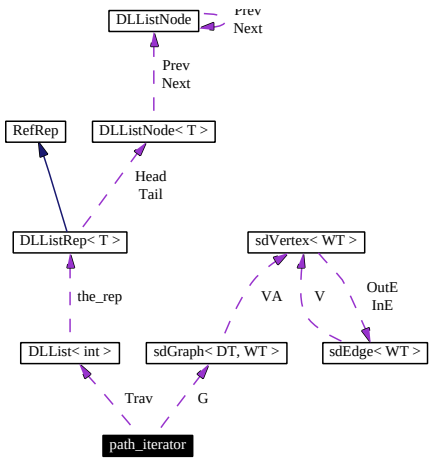
Definition at line 55 of file sdgiter.inc.

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

- **sdgiter.inc**

4.46 path_iterator Class Reference

Collaboration diagram for path_iterator:



Public Methods

- int **bad** () const
- int **source** () const
- int **target** () const
- int **length** () const
- void **copy** (const path_iterator &**pi**)
- const path_iterator& **operator=** (const path_iterator &**pi**)
- **path_iterator** (path_iterator &**pi**)
- **path_iterator** ()
- **~path_iterator** ()
- const path_iterator& **operator++** ()
- **operator int** () const
- int* **operator** * () const
- void **reset** ()

Private Methods

- **path_iterator** (const **sdGraph**< DT, WT > *sdg, int s, int t, int k, **DLList**< int > *const *tl, int(*filter)(const **sdGraph**< DT, WT > &graph, const int *path, int ndx, int pathlen, void *arbitrary_criteria), void *criteria)

Private Attributes

- const **sdGraph**<DT,WT>* **G**
- int **K**
- int* **Path**
- **DLList**<int>::**Iterator**** **Trav**
- int **D**
- int **Done**

- int **AlmostDone**
- int (∗ **Filter**)(const **sdGraph**< DT, WT > &,const int ∗,int, int, void ∗)
- void∗ **Criteria**

Friends

- class **sdGraph**< **DT**, **WT** >

4.46.1 Constructor & Destructor Documentation

4.46.1.1 path_iterator::path_iterator (path_iterator & *pi*) [inline]

Definition at line 405 of file sdgiter.inc.

```
00406                   {
00407                     G = 0; K = -1; D = -1;
00408                     Done = 1; AlmostDone = 1;
00409                     Path = 0; Trav = 0;
00410                     Filter = 0;
00411                     Criteria = 0;
00412                     copy(pi);
00413
```

4.46.1.2 path_iterator::path_iterator () [inline]

Definition at line 415 of file sdgiter.inc.

```
00416                   {
00417                     G = 0; K = -1; D = -1;
00418                     Done = 1; AlmostDone = 1;
00419                     Path = 0; Trav = 0; Path = 0;
00420                     Filter = 0;
00421                     Criteria = 0;
00422
```

4.46.1.3 path_iterator::~path_iterator () [inline]

Definition at line 424 of file sdgiter.inc.

```
00425                   {
00426                     delete[] Path;
00427                     for(int i=0; i<K-1; ++i) delete[] Trav[i];
00428                     delete[] Trav;
00429
```

4.46.1.4 path_iterator::path_iterator (const sdGraph< DT, WT > ∗ *sdg*, int *s*, int *t*, int *k*, DLList< int > ∗const ∗ *tl*, int(∗ *filter*)(const sdGraph< DT, WT > &graph, const int ∗path, int ndx, int pathlen, void ∗arbitrary_criteria), void ∗ *criteria*) [inline, private]

Definition at line 562 of file sdgiter.inc.

```
00576                   {
00577                     int n = sdg->NumV;
00578                     G = 0; K = -1; D = -1; AlmostDone = Done = 1;
00579                     Path = 0; Trav = 0;
00580
00581                     if(k<0 || n<1 || s<0 || s>=n || t<0 || t>=n) return;
00582
00583                     if(!tl && k>1) return;
00584
00585                     if(!(Path = new int[k+1])) { return; }
00586
00587                     if(k>1)
00588                       {
00589                         if(!(Trav = new DLList<int>::Iterator*[k-1]))
00590                           { delete[] Path; return; }
00591
00592                         for(int i=0; i<k-1; ++i)
00593                           {
00594                             if(!(Trav[i] = new DLList<int>::Iterator[n]))
00595                               {
00596                                 for(int j=0; j<=i; ++j)
00597                                   { delete[] Trav[j]; }
00598                                 delete[] Path; delete[] Trav;
00599                                 return;
00600                               }
00601                             for(int j=0; j<n; ++j)
00602                                 Trav[i][j] = tl[i][j].GetIterator();
00603                           }
00604                       }
00605
```

4.46.2 Member Function Documentation

4.46.2.1 int path_iterator::bad () const [inline]

Definition at line 328 of file sdgiter.inc.

Referenced by copy().

```
00329                   { return (!G || !Path || (K>1 && !Trav)); }
00330
```

4.46.2.2 void path_iterator::copy (const path_iterator & *pi*) [inline]

Definition at line 335 of file sdgiter.inc.

Referenced by operator=(), and path_iterator().

```
00336                   {
00337                     if(this == &pi) return;
00338
00339                     if(pi.bad())
00340                       // If "pi" is bad then we make this iterator bad (copy
00341                       // a bad iterator to a good iterator makes the good
00342                       // iterator bad!).
00343                       {
00344                         for(int i=0; i<K-1; ++i) delete[] Trav[i];
00345                         delete[] Trav; delete[] Path;
00346                         G=0; K=0; D=-1; Done=1;
00347                         return;
00348                       }
```

```
00349
00350                     DLList<int>::Iterator ** tmpT = 0;
00351                     int * tmpP = 0;
00352
00353                     if(G->NumV != pi.G->NumV || K!=pi.K)
00354                       // Have to resize our lists.
00355                       {
00356                         tmpT = Trav; tmpP = Path;
00357
00358                         if(!(Path = new int[pi.K+1]))
00359                             { Path = tmpP; return; }
00360
00361                         if(pi.K>1)
00362                           {
00363                             if(!(Trav = new DLList<int>::Iterator*[pi.K-1]))
00364                               {
00365                                 delete[] Path; delete[] Trav;
00366                                 Path = tmpP; Trav = tmpT;
00367                                 return;
00368                               }
00369                             for(int i=0; i<pi.K-1; ++i)
00370                               {
00371                          if(!(Trav[i] = new DLList<int>::Iterator[pi.G->NumV]))
00372                                   {
00373                                     for(int j=0; j<i; ++j) delete[] Trav[j];
00374                                     delete[] Path; delete[] Trav;
00375                                     Path = tmpP; Trav = tmpT;
00376                                     return;
00377                                   }
00378                               }
00379                           }
00380                         else Trav = 0;
00381
00382                         if(K>1) for(int i=0; i<K-1; ++i) delete[] tmpT[i];
00383                         delete[] tmpP; delete[] tmpT;
00384                         G = pi.G; K = pi.K;
00385                       }
00386
00387                     for(int i=0; i<K-1; ++i)
00388                       {
00389                         for(int j=0; j<G->NumV; ++j)
00390                             Trav[i][j] = pi.Trav[i][j];
00391                         Path[i+1] = pi.Path[i+1];
00392                       }
00393
```

4.46.2.3 int path_iterator::length () const [inline]

Definition at line 333 of file sdgiter.inc.

```
00333                                 { return K; }
00334
```

4.46.2.4 int * path_iterator::operator * () const [inline]

Definition at line 536 of file sdgiter.inc.

```
00537                   {
00538                     int * current_path;
00539                     if(!(current_path = new int[K+1])) return 0;
```

```
00540                     for(int i=0; i<K+1; ++i) current_path[i] = Path[i];
00541                     return current_path;
00542
```

4.46.2.5 path_iterator::operator int () const [inline]

Definition at line 534 of file sdgiter.inc.

```
00534                                     { return !Done; }
00535
```

4.46.2.6 const path_iterator & path_iterator::operator++ () [inline]

Definition at line 431 of file sdgiter.inc.

```
00432                   {
00433
00434                     int found, pre;
00435
00436                     if(bad()) AlmostDone = 1;
00437                     if(AlmostDone) Done = 1;
00438                     if(Done) return *this;
00439
00440                     if(K==0)
00441                       {
00442 #ifdef DEBUG
00443   ++PathsVisited;
00444 #endif
00445                         AlmostDone = 1;
00446                         if(D != Path[0]) Done = 1;
00447                         return *this;
00448                       }
00449
00450                     if(K==1)
00451                       {
00452 #ifdef DEBUG
00453   ++PathsVisited;
00454 #endif
00455                         if(G->HasEdge(Path[0],Path[1]) &&
00456                           ((!Filter || !Filter(*G,Path,0,1,Criteria))))
00457                             AlmostDone=1;
00458                         else { AlmostDone = Done = 1; }
00459                         return *this;
00460                       }
00461
00462                     // Backward stage of the iteration: build up a path
00463                     // from the end to the beginning.
00464                     //
00465                     for(; D>=0; --D)
00466                       {
00467                         pre = Path[D+2];
00468
00469                         if(!Trav[D][pre])
00470                           // If this happens it means no path exists.
00471                           { AlmostDone = Done = 1; return *this; }
00472
00473                         for(found = 0; Trav[D][pre]; ++(Trav[D][pre]))
00474                           {
00475                             if(found) break;
00476
```

```
00477                             Path[D+1] = *(Trav[D][pre]);
00478
00479 #ifdef DEBUG
00480   ++PathsVisited;
00481 #endif
00482
00483                             // Apply the filter to this path.  If the
00484                             // path is to be filtered, we assume all
00485                             // paths containing the current path as a
00486                             // subset should not be expanded.  If this
00487                             // path should be filtered, we simply skip
00488                             // this and look for a different path.
00489                             //
00490                             if(!Filter || !Filter(*G,Path,D+1,K,Criteria))
00491                                 found = 1;
00492                           }
00493
00494                         if(!found)
00495                           // All the sub-paths got filtered so we need to back
00496                           // up to find another candidate sub-path.
00497                           {
00498                             for(; D<K-1; ++D)
00499                               {
00500                                 pre = Path[D+2];
00501                                 if(!Trav[D][pre]) Trav[D][pre].reset();
00502                                 else break;
00503                               }
00504
00505                             // If we're at the beginning, then we must be
00506                             // done.
00507                             if(D==K-1)
00508                               { AlmostDone = Done = 1; return *this; }
00509                             else ++D;
00510                           }
00511                       }
00512
00513                     // Forward stage of the iteration: prepare the matrices
00514                     // for the next iteration.  We reset all the matrices/sets
00515                     // that are *completely* empty.  As soon as we reach a set
00516                     // that contains elements in it we stop.  This is the point
00517                     // where the next search will pick up.
00518
00519                     for(D=0; D<K-1; ++D)
00520                       {
00521                         pre = Path[D+2];
00522                         if(!Trav[D][pre]) Trav[D][pre].reset();
00523                         else break;
00524                       }
00525
```

4.46.2.7 const path_iterator & path_iterator::operator= (const path_iterator & *pi*) [inline]

Definition at line 402 of file sdgiter.inc.

```
00403                 { copy(pi); return *this; }
00404
```

4.46.2.8 void path_iterator::reset () [inline]

Definition at line 544 of file sdgiter.inc.

```
00545                   {
00546                     if(K>1)
00547                       {
00548                         for(; D<K-1; ++D)
00549                             for(int i=0; i<G->NumV; ++i)
00550                                 Trav[D][i].reset();
00551                         D = K-2;
00552                       }
00553
```

4.46.2.9 int path_iterator::source () const [inline]

Definition at line 331 of file sdgiter.inc.

```
00331                                   { return Path ? Path[0] : -1; }
00332
```

4.46.2.10 int path_iterator::target () const [inline]

Definition at line 332 of file sdgiter.inc.

```
00332                                   { return Path ? Path[K] : -1; }
00333
```

4.46.3 Friends And Related Function Documentation

4.46.3.1 class sdGraph [friend]

Definition at line 326 of file sdgiter.inc.

4.46.4 Member Data Documentation

4.46.4.1 int path_iterator::AlmostDone [private]

Definition at line 627 of file sdgiter.inc.

4.46.4.2 void * path_iterator::Criteria [private]

Definition at line 633 of file sdgiter.inc.

4.46.4.3 int path_iterator::D [private]

Definition at line 624 of file sdgiter.inc.

4.46.4.4 int path_iterator::Done [private]

Definition at line 625 of file sdgiter.inc.

4.46.4.5 int (* path_iterator::Filter)(const sdGraph<DT,WT>&,const int*,int,int,void*) [private]

Referenced by operator++().

4.46.4.6 const sdGraph< DT, WT > * path_iterator::G [private]

Definition at line 618 of file sdgiter.inc.

4.46.4.7 int path_iterator::K [private]

Definition at line 619 of file sdgiter.inc.

4.46.4.8 int * path_iterator::Path [private]

Definition at line 620 of file sdgiter.inc.

4.46.4.9 DLList< int >::Iterator ** path_iterator::Trav [private]

Definition at line 621 of file sdgiter.inc.

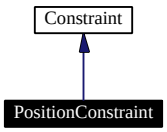
The documentation for this class was generated from the following file:

- **sdgiter.inc**

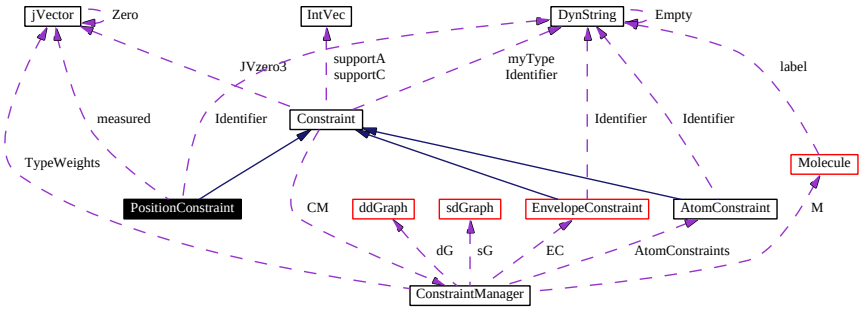
4.47 PositionConstraint Class Reference

`#include <position_constraint.h>`

Inheritance diagram for PositionConstraint



Collaboration diagram for PositionConstraint:



Public Methods

- **PositionConstraint** (**ConstraintManager** &cm)
- **PositionConstraint** (**ConstraintManager** &cm, int atom_i, double x, double y, double **z**, double variance_in)
- **~PositionConstraint** ()
- **jVector dStress** (int k, const **jVector** &pos) const
- int **Compare** (const PositionConstraint &dc) const
- **MAKECOMPARISONS_WITH_COMPARE** (PositionConstraint)
- int **Output** (ostream &os) const
- int **OutputBinary** (ostream &os) const

Static Public Attributes

- **DynString Identifier** = "atompos:"

Protected Methods

- double **computeStress** () const
- double **computeStress** (int k, const **jVector** &pos) const

  Mimics moving the atom to a new position.

- int **Read** (istream &is)
- int **ReadBinary** (istream &is)

  Same scenerio as **Read***()* (p. 388) *only binary version.*

- void **PreSetup** ()

Protected Attributes

- **jVector measured**

  input, measured position (the correct position).

- double **variance**
- double **stdev**

4.47.1 Detailed Description

//

position_constraint.h

This is a constraint type for GRAPE. It constrains the position of a single atom to set coordinates, each with a specified variance.

//

Definition at line 20 of file position_constraint.h.

4.47.2 Constructor & Destructor Documentation

4.47.2.1 PositionConstraint::PositionConstraint (ConstraintManager & *cm*)

Definition at line 3 of file position_constraint.C.

```
00004   : Constraint(cm,IntVec(1,-1))
00005   {
00006     // this is an unititialized constraint.  The atom position
00007     // and the variance values must be added.
00008     measured[0]=0;
00009     measured[1]=0;
00010     measured[2]=0;
00011     //    variance[0] = INF;
00012     //    variance[1] = INF;
00013     //    variance[2] = INF;
00014     variance = INF;
00015
```

4.47.2.2 PositionConstraint::PositionConstraint (ConstraintManager & *cm*, int *atom_i*, double *x*, double *y*, double *z*, double *variance_in*)

Definition at line 20 of file position_constraint.C.

```
00023   : Constraint(cm, IntVec(1,atom_i))
00024
00025 {
00026   measured[0]=x;
00027   measured[1]=y;
00028   measured[2]=z;
00029   //  variance[0] = x_var;
```

```
00030   //variance[1] = y_var;
00031   //variance[2] = z_var;
00032   variance = variance_in;
00033   //  stdev[0] = sqrt(x_var);
00034   //  stdev[1] = sqrt(y_var);
00035   //  stdev[2] = sqrt(z_var);
00036   stdev = sqrt(variance);
00037
```

4.47.2.3 PositionConstraint::~PositionConstraint () [inline]

Definition at line 34 of file position_constraint.h.

```
00034                       {}
00035
```

4.47.3 Member Function Documentation

4.47.3.1 int PositionConstraint::Compare (const PositionConstraint & *pc*) const [inline]

Definition at line 72 of file position_constraint.h.

```
00073 {
00074   if(supportA[0] == pc.supportA[0])
00075     return 0;
00076   if(supportA[0]<pc.supportA[0])
00077     return -1;
00078   return 1;
00079
```

4.47.3.2 PositionConstraint::MAKECOMPARISONS_WITH_COMPARE (PositionConstraint)

4.47.3.3 int PositionConstraint::Output (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 137 of file position_constraint.C.

```
00138 {
00139   os << supportA[0] << ' ' << measured[0] << ' '<< measured[1] << ' '
00140      << measured[2] << ' ' << variance;
00141   return 2;
00142
```

4.47.3.4 int PositionConstraint::OutputBinary (ostream & *os*) const [virtual]

Reimplemented from **Constraint** (p. 84).

Definition at line 151 of file position_constraint.C.

```
00152 {
00153   os <= supportA <= measured <= variance;
00154   return 2;
00155
```

4.47.3.5 void PositionConstraint::PreSetup (void) [protected, virtual]

The setup function is called if the uninialized constraint method is used to create the constraint to allow different data representations within the constraint. Without this, only data read in from the input file will be available to the member functions. For simple constraints, this is really not very useful. However for multi-atom constraints that need to be a little 'smarter' about data representations, the Setup function will be helpful.

Reimplemented from **Constraint** (p. 85).

Definition at line 39 of file position_constraint.C.

```
00040 {
00041   //  stdev[0] = sqrt(variance[0]);
00042   //  stdev[1] = sqrt(variance[1]);
00043   //  stdev[2] = sqrt(variance[2]);
00044   stdev = sqrt(variance);
00045
```

4.47.3.6 int PositionConstraint::Read (istream & *is*) [protected, virtual]

Derived classes need to define here how to read themselves in. Notice that the base class defines **Input**() (p. 83) as a wrapper around this virtual function. Subclasses should typically overwrite just the **Read**() (p. 388) function (and not the **Input**() (p. 83) function).

In the **Read**() (p. 388) function, the derived class should *not* log itself with the constraint manager since this is taken care of in the base classes **Input**() (p. 83) function.

Reimplemented from **Constraint** (p. 85).

Definition at line 144 of file position_constraint.C.

```
00145 {
00146   is >> supportA[0] >> measured[0] >>  measured[1]
00147      >> measured[2] >> variance;
00148   return 2;
00149
```

4.47.3.7 int PositionConstraint::ReadBinary (istream & *is*) [protected, virtual]

Same scenerio as **Read**() (p. 388) only binary version.

Reimplemented from **Constraint** (p. 85).

Definition at line 157 of file position_constraint.C.

```
00158 {
00159   is >= supportA[0] >= measured >= variance;
00160   return 2;
00161
```

4.47.3.8 double PositionConstraint::computeStress (int *k*, const jVector & *pos_k*) const [protected, virtual]

Mimics moving the atom to a new position.

Reimplemented from **Constraint** (p. 89).

Definition at line 70 of file position_constraint.C.

```
00071 {
00072
00073   if(k != supportA[0])
00074     {
00075       cerr << "Fatal error: position_constraint.C, computeStress("<< k \
00076            <<", pos) not legal on position constraint for atom " \
00077            << supportA[0] << endl;
00078       DIE("");
00079     }
00080
```

4.47.3.9 double PositionConstraint::computeStress () const [protected, virtual]

Returns how similar the distance between atoms i and j in molecule, mol, are compared to the measured distance represented in this constraint.

Reimplemented from **Constraint** (p. 89).

Definition at line 52 of file position_constraint.C.

```
00053 {
00054
00055
00056   // Find the current position in the predicted molecule:
00057   jVector diff = getMolecule()(supportA[0])-measured;
00058
00059   // Return the squared difference between this and our model position:
00060   // double calc = (diff[0]*diff[0]/variance[0])+
00061   // (diff[1]*diff[1]/variance[1])+
00062   // (diff[2]*diff[2]/variance[2]);
00063   double calc = (diff[0]*diff[0]+ \
00064                  (diff[1]*diff[1])+\
00065                  (diff[2]*diff[2]));
00066
00067   return calc/(2.0*variance);
00068
```

4.47.3.10 jVector PositionConstraint::dStress (int *k*, const jVector & *pos_k*) const [virtual]

asking constraint number k for its 12 element dStress vector we pass in pos_a vector – corresponding to the atom position if the constraint is an atom, and corresponding to the position of the atom k in question if the call is to a non-atom constraint. EG for a distance constraint atom 12 needs the dStress from the d-constraint, so it will call dStress on the d-constraint(12, atom_pos) so the d_constr can answer the question if atom 12 were in atom_pos, I would say my dStress is blah.

Reimplemented from **Constraint** (p. 89).

Definition at line 95 of file position_constraint.C.

```
00096 {
00097   // A HACK and a TEMPORARY FIX to an inherent logic problems in this
00098   // set of functions.  come see me w/ questions -jd
00099
00100
00101   if(k != supportA[0])
00102     {
00103       cerr << "Fatal error: position_constraint.C, dStress("<< k \
00104            <<", pos) not legal on position constraint for atom " <<
00105         supportA[0] << endl;
```

```
00106       DIE("");
00107     }
00108
```

4.47.4 Member Data Documentation

4.47.4.1 DynString PositionConstraint::Identifier = "atompos:" [static]

Reimplemented from **Constraint** (p. 95).

Definition at line 118 of file globals.C.

4.47.4.2 jVector PositionConstraint::measured [protected]

input, measured position (the correct position).

Definition at line 59 of file position_constraint.h.

4.47.4.3 double PositionConstraint::stdev [protected]

Definition at line 62 of file position_constraint.h.

4.47.4.4 double PositionConstraint::variance [protected]

Definition at line 61 of file position_constraint.h.

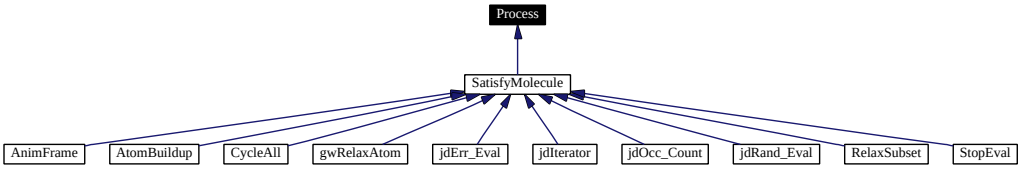
The documentation for this class was generated from the following files:

- **position_constraint.h**
- **globals.C**
- **position_constraint.C**

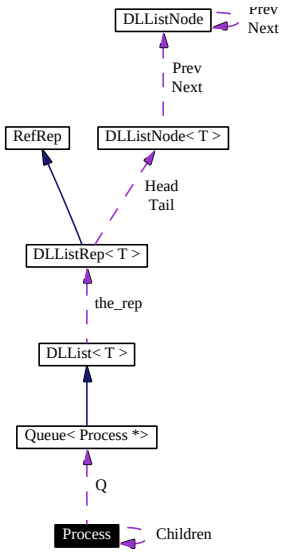
4.48 Process Class Reference

`#include <process.h>`

Inheritance diagram for Process



Collaboration diagram for Process:



Public Methods

- virtual **~Process** ()
- virtual int **run** ()=0
- virtual void **reset** ()

  Run any required operations to reset the process.

- void **runChildren** ()
- void **runChildrenRepeatFirst** ()
- int **runChild** (int i)
- int **runNextChild** ()
- void **addChild** (Process *c=NULL)
- int **numChildren** () const
- int **numQueued** () const

  Returns the number of children awaiting execution.

- void **cleanQueue** ()

Removes all queued jobs without run' ing them.

- void **setPriority** (double pri)

  Sets the priority of the process (low priorities are executed first.

- double **getPriority** () const

  Returns the priority of this process.

- int **Compare** (const Process &p) const

  Compares processes based on their priority.

- **MAKECOMPARISONS_WITH_COMPARE** (Process)

  Give me all the comparison operators.

Protected Methods

- **Process** (double pri=-INF, int numChildren=0)

Protected Attributes

- double **Priority**

  High priorities are executed first. These can range from -INF to +INF.

- Process** **Children**

  An array of pointers to sub-processes.

- int **numC**

  The number of total children (including zombies).

- bool **active**

  defaults to t, if f, the process is never run.

Private Methods

- void **assertOrder** ()

Private Attributes

- int **sorted**
- **Queue**<Process *> **Q**

4.48.1 Detailed Description

//

File: Process.h

Abstract class to manage and provide functionality for a set of 'jobs.' Each job inherits from class Process and can create children, add them to a queue and execute chilren on that queue. All jobs must have a member called **run**(void) (p. 396).

//

Definition at line 24 of file process.h.

4.48.2 Constructor & Destructor Documentation

4.48.2.1 Process::~Process () [virtual]

Definition at line 18 of file process.C.

```
00019 {
00020   if(Children)
00021     {
00022       for(int i=0; i<numC; ++i)
00023         delete Children[i];
00024       delete[] Children;
00025       Children=0;
00026     }
00027
```

4.48.2.2 Process::Process (double *pri* = -INF, int *numChildren* = 0) [protected]

Derived classes can construct a process with a specified priority and a specified number of children. These Children are *not* added to the queue.

Definition at line 4 of file process.C.

```
00005 {
00006   Children=0;
00007   numC = 0;
00008   sorted = 0;
00009
00010   if(numChildren>0 && (Children = new Process*[numChildren]))
00011     {
00012       for(int i=0; i<numChildren; i++)
00013         Children[i] = 0;
00014       numC = numChildren;
00015     }
00016
```

4.48.3 Member Function Documentation

4.48.3.1 int Process::Compare (const Process & *p*) const [inline]

Compares processes based on their priority.

Definition at line 99 of file process.h.

```
00101   {
00102     if(Priority<p.Priority)
00103       return -1;
00104     else if(Priority>p.Priority)
00105       return 1;
00106     return 0;
00107
```

4.48.3.2 Process::MAKECOMPARISONS_WITH_COMPARE (Process)

Give me all the comparison operators.

4.48.3.3 void Process::addChild (Process * p = NULL)

Adds a child to the execution queue. Note that processes do not need to queue children from the Children array. For example, a process may add itself to the execution queue for recursive purposes (via addChild(this)).

Definition at line 160 of file process.C.

Referenced by RelaxSubset::run(), and CycleAll::run().

```
00161 {
00162   // If this guy's priority is not extremal, we'll have to resort
00163   // at some point in the future.  If it *is* extremal, we just tack
00164   // this process is on one of the queue's ends.
00165   if(Q && p->Priority <= Q.PeekFront()->Priority)
00166     Q.AddToFront(p);
00167   else if(Q && p->Priority >= Q.PeekBack()->Priority)
00168     Q.AddToBack(p);
00169   else
00170     {
00171       Q.AddToBack(p);
00172       sorted = 0;
00173     }
00174
```

4.48.3.4 void Process::assertOrder () [private]

Definition at line 29 of file process.C.

Referenced by runChild(), runChildren(), runChildrenRepeatFirst(), and runNextChild().

```
00030 {
00031   if (sorted) return;
00032   Process ** child;
00033   int numQ = Q.Number();
00034   if (numQ>0)
00035     {
00036       // Empty out the queue and then sort all the children
00037       // that were in there.
00038       if(!(child = new Process*[numQ]))
00039         DIE("Process::assertOrder memory failure");
00040
00041       for(int i=0; Q; ++i)
00042         child[i] = Q.RemoveFromBack();
00043
00044       QuickSortPtrs(child, numQ);
00045
```

```
00046       // ACK!  REMOVE ME
00047 //         for(i=0; i<(nnonhelatoms-1); i++) {
00048 //         for(j=(nnonhelatoms-1); j>i; j--) {
00049 //           atom1 = atomorder[j-1];
00050 //           atom2 = atomorder[j];
00051 // //     debug_i ("SORT:atom1", atom1);
00052 // //     debug_i ("SORT:atom2", atom2);
00053 // //     debug_d ("SORT:a1err", atom_info[atom1].lfnc);
00054 // //     debug_d ("SORT:a2err", atom_info[atom2].lfnc);
00055 //        if(
00056 //           (fabs(atom_info[atom1].lfnc-atom_info[atom2].lfnc) < 1e-12) &&
00057 //           (atom1 > atom2)
00058 //           ){
00059 //          itemp = atomorder[j-1];
00060 //          atomorder[j-1] = atomorder[j];
00061 //          atomorder[j] = itemp;
00062 //        }
00063 //
00064 //      }
00065 //       }
00066 //
00067
00068       // for equal priority, sort by atom index
00069 //       Process * temp;
00070 //       int atom1, atom2;
00071 //       for(int i=0; i<numQ-1; i++) {
00072 //      for (int j = numQ-1; j > i;j--){
00073 //        atom1 = ((gwRelaxAtom *) child[j-1])->atom_index; // ALSO FIX THIS IS NOW PUBLIC
00074 //        atom2 = ((gwRelaxAtom *) child[j])->atom_index;
00075 //        if(
00076 //           (fabs(child[j-1]->Priority - child[j]->Priority) < 1e-12) &&
00077 //           (atom1 < atom2)
00078 //           ){
00079 //          temp = child[j-1];
00080 //          child[j-1] = child[j];
00081 //          child[j] = temp;
00082 //        }
00083 //
00084 //        }
00085 //      }
00086
00087       // Put the sorted list back into the queue.
00088       for(int i=0; i<numQ; ++i)
00089         Q.Add(child[i]);
00090
00091       // Get rid of the extra array of pointers.
00092       delete[] child;
00093
00094       sorted = 1;
00095     }
00096
```

4.48.3.5 void Process::cleanQueue ()

Removes all queued jobs without run' ing them.

Definition at line 136 of file process.C.

Referenced by RelaxSubset::run().

```
00137 {
00138   while(Q)
00139     Q.RemoveFromBack();
```

```
00140   // NB not deleted as in runChildren()
00141
```

4.48.3.6 double Process::getPriority () const [inline]

Returns the priority of this process.

Definition at line 156 of file process.h.

Referenced by gwRelaxAtom::run().

```
00157 { return Priority; }
00158
```

4.48.3.7 int Process::numChildren () const [inline]

Returns the *total* number of children created by this process. These children are not necessarily queued to execute. Call **numQueued**() (p. 396) if would like to find the number of children awaiting execution.

Definition at line 150 of file process.h.

```
00151 { return numC; }
00152
```

4.48.3.8 int Process::numQueued () const [inline]

Returns the number of children awaiting execution.

Definition at line 153 of file process.h.

```
00154 { return Q.Number(); }
00155
```

4.48.3.9 void Process::reset () [inline, virtual]

Run any required operations to reset the process.

Definition at line 36 of file process.h.

```
00036                       {}
00037
```

4.48.3.10 int Process::run (void) [pure virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented in **AtomBuildup** (p. 51), **CycleAll** (p. 120), **gwRelaxAtom** (p. 274), **AnimFrame** (p. 41), **jdErr_Eval** (p. 312), **jdIterator** (p. 316), **jdOcc_Count** (p. 319), **jdRand_Eval** (p. 322), **RelaxSubset** (p. 420), and **StopEval** (p. 493).

Referenced by main(), and runChildrenRepeatFirst().

4.48.3.11 int Process::runChild (int *i*)

Run the child with the ith smallest priority. This child is *not* removed from the queue! Returns the exit status of the child.

Definition at line 143 of file process.C.

```
00144 {
00145   if(i<0 || i>=Q.Number())
00146     return -1;
00147
00148   // Make sure the queue is ordered.
00149   assertOrder();
00150
00151   return Q.Get(i)->run();
00152
```

4.48.3.12 void Process::runChildren ()

Execute all the children that have been added to the queue by order of their priority. After the call, the queue of children is empty.

Definition at line 117 of file process.C.

```
00118 {
00119   // Make sure the queue is ordered.
00120   assertOrder();
00121
00122 //   Process * P;
00123 //   while(Q){
00124 //     P = Q.RemoveFromBack();
00125 //     if (P->getPriority() >= 0)
00126 //       P->run();
00127 //   }
00128
```

4.48.3.13 void Process::runChildrenRepeatFirst ()

Execute all the children that have been added to the queue by order of their priority. Repeat the first process at the end. After the call, the queue of children is empty.

Definition at line 98 of file process.C.

Referenced by CycleAll::run().

```
00099 {
00100   // Make sure the queue is ordered.
00101   assertOrder();
00102
00103    Process * P;
00104    if (Q)
00105      P = Q.RemoveFromBack();
00106    else
00107      return;
00108
00109    P->run();
00110    while (Q)
00111      Q.RemoveFromBack()->run();
00112    P->run();
00113
```

```
00114
00115
```

4.48.3.14 int Process::runNextChild ()

Runs the child with the next smallest priority. This child is removed from the queue and lost. Returns the exit status of the child.

Definition at line 154 of file process.C.

Referenced by RelaxSubset::run().

```
00155 {
00156   assertOrder();
00157   return Q.RemoveFromBack()->run();
00158
```

4.48.3.15 void Process::setPriority (double *pri*) [inline]

Sets the priority of the process (low priorities are executed first.

Definition at line 159 of file process.h.

```
00160 { Priority = pri; }
00161
```

4.48.4 Member Data Documentation

4.48.4.1 Process ** Process::Children [protected]

An array of pointers to sub-processes.

Definition at line 126 of file process.h.

4.48.4.2 double Process::Priority [protected]

High priorities are executed first. These can range from -INF to +INF.

Definition at line 122 of file process.h.

4.48.4.3 Queue< Process *> Process::Q [private]

Definition at line 142 of file process.h.

4.48.4.4 bool Process::active [protected]

defaults to t, if f, the process is never run.

Definition at line 132 of file process.h.

4.48.4.5 int Process::numC [protected]

The number of total children (including zombies).

Definition at line 129 of file process.h.

4.48.4.6 int Process::sorted [private]

Definition at line 139 of file process.h.

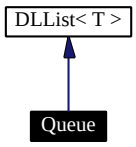
The documentation for this class was generated from the following files:

- **process.h**
- **process.C**

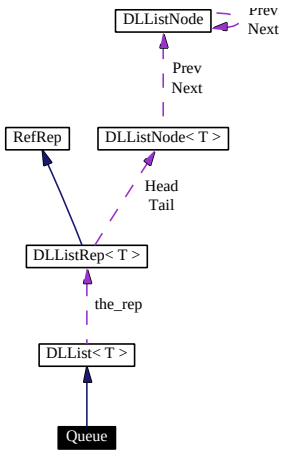
4.49 Queue Class Template Reference

`#include <queue.h>`

Inheritance diagram for Queue



Collaboration diagram for Queue:



Public Methods

- void **Add** (const T &x)
- void **Add** (const Queue< T > &q)
- T **Remove** ()
- const T& **Peek** () const
- T& **Peek** ()
- const Queue<T>& **operator+=** (const Queue< T > &q)

Friends

- Queue<T> **operator+** (Queue< T > x, const Queue< T > &y)

template<class T> class Queue

4.49.1 Member Function Documentation

4.49.1.1 template<class T> void Queue<T>::Add (const Queue< T > & *q*) [inline]

Definition at line 21 of file queue.h.

```
00022                   { AddToBack(q.List); }
00023
```

4.49.1.2 template<class T> void Queue<T>::Add (const T & x) [inline]

Reimplemented from **DLList** (p. 163).

Definition at line 19 of file queue.h.

Referenced by Process::assertOrder(), and bf_iterator::operator++().

```
00020                   { AddToBack(x); }
00021
```

4.49.1.3 template<class T> T & Queue<T>::Peek () [inline]

Definition at line 26 of file queue.h.

```
00026                           { return PeekFront(); }
00027
```

4.49.1.4 template<class T> const T & Queue<T>::Peek () const [inline]

Definition at line 25 of file queue.h.

```
00025                                   { return PeekFront(); }
00026
```

4.49.1.5 template<class T> T Queue<T>::Remove () [inline]

Reimplemented from **DLList** (p. 169).

Definition at line 23 of file queue.h.

Referenced by bf_iterator::operator++().

```
00024                   { return RemoveFromFront(); }
00025
```

4.49.1.6 template<class T> const Queue< T > & Queue<T>::operator+=<T> (const Queue< T > & q) [inline]

Definition at line 27 of file queue.h.

```
00028                   { Add(q); return *this; }
00029
```

4.49.2 Friends And Related Function Documentation

4.49.2.1 template<class T> Queue< T > operator+<T> (Queue< T > x, const Queue< T > & y) [friend]

Definition at line 29 of file queue.h.

```
00030                   { return x+=y; }
00031
```

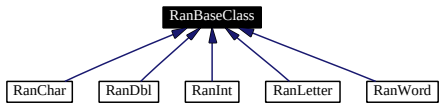
The documentation for this class was generated from the following file:

- **queue.h**

4.50 RanBaseClass Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanBaseClass



Public Methods

- virtual **~RanBaseClass** ()
- void **Seed** (**uint** seed_val)

Protected Methods

- **RanBaseClass** ()
- double **GetReal** () const

4.50.1 Constructor & Destructor Documentation

4.50.1.1 RanBaseClass::~RanBaseClass () [virtual]

Definition at line 113 of file randomgenerator.C.

```
00114   {}
00115
```

4.50.1.2 RanBaseClass::RanBaseClass () [protected]

Definition at line 107 of file randomgenerator.C.

```
00108   {
00109     if(!z.seeded)
00110         Seed();
00111
```

4.50.2 Member Function Documentation

4.50.2.1 double RanBaseClass::GetReal () const [protected]

Definition at line 116 of file randomgenerator.C.

Referenced by RanLetter::Get(), RanChar::Get(), RanInt::Get(), and RanDbl::Get().

```
00117   {
00118     double uni = z.u[z.i97] - z.u[z.j97];
00119     if(uni < 0.0) uni += 1.0;
00120     z.u[z.i97] = uni;
00121     z.i97 = (z.i97 == 1) ? 97 : z.i97-1;
```

```
00122     z.j97 = (z.j97 == 1) ? 97 : z.j97-1;
00123     z.c -= z.cd;
00124     if(z.c<0.0) z.c += z.cm;
00125     uni -= z.c;
00126     if(uni<0.0) uni += 1.0;
00127     return uni;
00128
```

4.50.2.2 void RanBaseClass::Seed (uint *seed_val* = 0)

Definition at line 11 of file randomgenerator.C.
Referenced by RanBaseClass().

```
00012   {
00013     uint ij,kl;
00014     int i,j,k,l,ii,jj,m;
00015     double s,t;
00016
00017
00018     if (seed_val == 0){
00019       // This is a self-seed procedure
00020       // Added 4/11/2000 -JD
00021       // This effectively makes 0 an invalid seed.
00022
00023 #ifdef WIN32
00024       while (seed_val == 0) {
00025
00026         // This sucks big-time: returns time in seconds since Jan 1, 1970.
00027         seed_val = (unsigned) time(0);
00028       }
00029 #else
00030       // NOT WIN32
00031       while (seed_val == 0) { // 0 is an invalid seed value
00032
00033         // This, of course, still sucks... but if we get interrupted in our read
00034         // from /dev/random below and don't get a full unsigned worth of random
00035         // bytes, we want to have *something* non-zero filling up seed_val.
00036         seed_val = (unsigned)getpid() ^ (unsigned)time(0);
00037
00038 #ifdef HAVE_LIBPTHREAD
00039         // If we are using threads on a system w/o /dev/random, and if two threads
00040         // both try to generate one of these "magic seeds" at exactly the same
00041         // time, we don't want them both getting identical rngs.
00042         seed_val ^= (unsigned)pthread_self();
00043 #endif
00044
00045         //#ifdef linux
00046
00047         // I love /dev/random... but if we are on a system that doesn't support
00048         // it, this open will fail and we'll have to make due with the crappy
00049         // "randomness" from the above.
00050         int fd = open("/dev/random", O_RDONLY);
00051         if (fd != -1) {
00052           // Can we be interrupted in this read?  I guess so, so we'll settle for
00053           // some randomness, if not a full sizeof(s)-bytes worth...
00054           while (read(fd, &seed_val, sizeof(seed_val)) == 0);
00055           close(fd);
00056         }
00057         // #endif
00058
00059       }
00060
```

```
00061 #endif
00062
00063     } // end self-seed procedure
00064
00065
00066     // ij is the first seed to the random generator and kl is the
00067     // second seed.  These seeds are derived from the bits contained
00068     // in the number seed.  Specifically, interleaved bits are
00069     // assigned to each seed and then the bits are reversed (making the
00070     // lower order bits more significant.
00071     //
00072     ij = reverse_bits( 0xAAAAAAAA & seed_val );
00073     kl = reverse_bits( 0x77777777 & seed_val );
00074     ij = (uint) (31328 * ((double) ij / (double) UINT_MAX));
00075     kl = (uint) (30081 * ((double) kl / (double) UINT_MAX));
00076
00077     i = (ij/177)%177+2;
00078     j = ij%177 + 2;
00079     k = (kl/169)%178 + 1;
00080     l = kl%169;
00081
00082     for(ii=1; ii<=97; ii++)
00083       {
00084         s = 0.0;
00085         t = 0.5;
00086         for(jj=1; jj<=24; jj++)
00087           {
00088             m = (((i*j)%179)*k) % 179;
00089             i = j;
00090             j = k;
00091             k = m;
00092             l = (53*1 + 1) % 169;
00093             if((l*m)%64 >= 32) s+= t;
00094             t *= 0.5;
00095           }
00096         z.u[ii] = s;
00097       }
00098
```

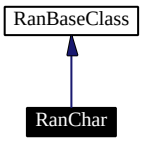
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

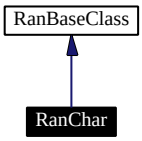
4.51 RanChar Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanChar



Collaboration diagram for RanChar:



Public Methods

- **RanChar** ()
- **~RanChar** ()
- char **Get** () const

4.51.1 Constructor & Destructor Documentation

4.51.1.1 RanChar::RanChar () [inline]

Definition at line 122 of file randomgenerator.h.

```
00122                          {}
00123
```

4.51.1.2 RanChar::~RanChar () [inline]

Definition at line 123 of file randomgenerator.h.

```
00123                           {}
00124
```

4.51.2 Member Function Documentation

4.51.2.1 char RanChar::Get () const

Definition at line 153 of file randomgenerator.C.

```
00154   {
00155     int index = (int) (95.0 * GetReal());
00156     return (' ' + (char) index);
00157
```

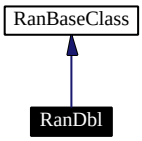
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

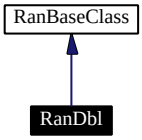
4.52 RanDbl Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanDbl



Collaboration diagram for RanDbl:



Public Methods

- **RanDbl** (double minimum=0.0, double maximum=1.0)
- **~RanDbl** ()
- double **Get** () const
- double **Get** (double minimum, double maximum) const

Private Attributes

- double **min**
- double **max**

4.52.1 Constructor & Destructor Documentation

4.52.1.1 RanDbl::RanDbl (double *minimum* = 0.0, double *maximum* = 1.0)

Definition at line 130 of file randomgenerator.C.

```
00131   { min = minimum; max = maximum; }
00132
```

4.52.1.2 RanDbl::~RanDbl () [inline]

Definition at line 93 of file randomgenerator.h.

```
00093                         {}
00094
```

4.52.2 Member Function Documentation

4.52.2.1 double RanDbl::Get (double *minimum*, double *maximum*) const

Definition at line 136 of file randomgenerator.C.

```
00137   { return (minimum + (maximum-minimum)*GetReal()); }
00138
```

4.52.2.2 double RanDbl::Get () const

Definition at line 133 of file randomgenerator.C.
Referenced by Envelope::Build().

```
00134   { return (min + (max-min)*GetReal()); }
00135
```

4.52.3 Member Data Documentation

4.52.3.1 double RanDbl::max [private]

Definition at line 97 of file randomgenerator.h.

4.52.3.2 double RanDbl::min [private]

Definition at line 97 of file randomgenerator.h.

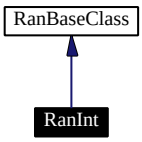
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

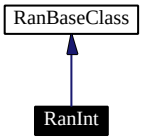
4.53 RanInt Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanInt



Collaboration diagram for RanInt:



Public Methods

- **RanInt** (int minimum=0, int maximum=1)
- **~RanInt** ()
- int **Get** () const
- int **Get** (int minimum, int maximum) const

Private Attributes

- int **min**
- int **max**

4.53.1 Constructor & Destructor Documentation

4.53.1.1 RanInt::RanInt (int *minimum* = 0, int *maximum* = 1)

Definition at line 139 of file randomgenerator.C.

```
00140   {
00141     if(minimum <= maximum)
00142       { min = minimum; max = maximum; }
00143     else
00144       { min = maximum; max = minimum; }
00145
```

4.53.1.2 RanInt::~RanInt () [inline]

Definition at line 109 of file randomgenerator.h.

```
00109                         {}
00110
```

4.53.2 Member Function Documentation

4.53.2.1 int RanInt::Get (int *minimum*, int *maximum*) const

Definition at line 150 of file randomgenerator.C.

```
00151   { return( minimum + (int) (((double) (maximum-minimum+1)) * GetReal()) ); }
00152
```

4.53.2.2 int RanInt::Get () const

Definition at line 147 of file randomgenerator.C.
Referenced by RanWord::Get(), and Molecule::setRandom().

```
00148   { return( min + (int) (((double) (max-min+1)) * GetReal()) ); }
00149
```

4.53.3 Member Data Documentation

4.53.3.1 int RanInt::max [private]

Definition at line 116 of file randomgenerator.h.

4.53.3.2 int RanInt::min [private]

Definition at line 116 of file randomgenerator.h.

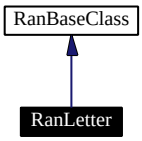
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

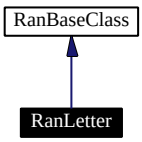
4.54 RanLetter Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanLetter



Collaboration diagram for RanLetter:



Public Methods

- **RanLetter** ()
- **~RanLetter** ()
- char **Get** () const

4.54.1 Constructor & Destructor Documentation

4.54.1.1 RanLetter::RanLetter () [inline]

Definition at line 132 of file randomgenerator.h.

```
00132                           {}
00133
```

4.54.1.2 RanLetter::~RanLetter () [inline]

Definition at line 133 of file randomgenerator.h.

```
00133                            {}
00134
```

4.54.2 Member Function Documentation

4.54.2.1 char RanLetter::Get () const

Definition at line 159 of file randomgenerator.C.
Referenced by RanWord::Get().

```
00160   {
00161     int index = (int) (52.0 * GetReal());
00162     if(index < 26)
00163         return ('A' + (char) index);
00164     else
00165         return ('a' + (char) (index-26));
00166
```

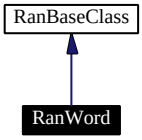
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

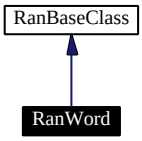
4.55 RanWord Class Reference

`#include <randomgenerator.h>`

Inheritance diagram for RanWord



Collaboration diagram for RanWord:



Public Methods

- **RanWord** ()
- **~RanWord** ()
- **DynString Get** (int exact_length) const
- **DynString Get** (int min_length=1, int max_length=100) const

4.55.1 Constructor & Destructor Documentation

4.55.1.1 RanWord::RanWord () [inline]

Definition at line 141 of file randomgenerator.h.

```
00141                         {}
00142
```

4.55.1.2 RanWord::~RanWord () [inline]

Definition at line 142 of file randomgenerator.h.

```
00142                          {}
00143
```

4.55.2 Member Function Documentation

4.55.2.1 DynString RanWord::Get (int *min_length* = 1, int *max_length* = 100) const

Definition at line 168 of file randomgenerator.C.

```
00169   {
00170     int length;
00171     RanInt poof_a_number;
00172     RanLetter poof_a_letter;
00173 
00174     if(min_length<1)
00175         min_length = 1;
00176     if(max_length<min_length)
00177         max_length = min_length;
00178 
00179     if(min_length!=max_length)
00180         length = poof_a_number.Get(min_length, max_length);
00181     else
00182         length = min_length;
00183 
00184     DynString word;
00185     word.DestResizeToFit(length);
00186     for(int i=0; i<length; ++i)
00187         word[i] = poof_a_letter.Get();
00188     return word;
00189 
```

4.55.2.2 DynString RanWord::Get (int *exact_length*) const

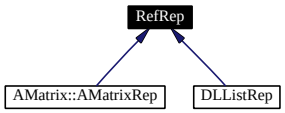
The documentation for this class was generated from the following files:

- **randomgenerator.h**
- **randomgenerator.C**

4.56 RefRep Class Reference

`#include <refrep.h>`

Inheritance diagram for RefRep



Public Methods

- **RefRep** ()
- **RefRep** (const RefRep &)
- virtual ~**RefRep** ()=0
- long **Reference** ()
- long **Dereference** ()
- long **References** () const

Private Attributes

- long **references**

4.56.1 Constructor & Destructor Documentation

4.56.1.1 RefRep::RefRep () [inline]

Definition at line 95 of file refrep.h.

```
00096 : references(1L)
00097
```

4.56.1.2 RefRep::RefRep (const RefRep &) [inline]

Definition at line 111 of file refrep.h.

```
00112 : references(1L)
00113
```

4.56.1.3 RefRep::~RefRep () [inline, pure virtual]

Definition at line 156 of file refrep.h.

```
00157 { /* empty */ }
00158
```

4.56.2 Member Function Documentation

4.56.2.1 long RefRep::Dereference () [inline]

Definition at line 134 of file refrep.h.

Referenced by ConstIterator::Copy(), Iterator::Copy(), DLList::operator=(), AMatrix::operator=(), AMatrix::~AMatrix(), ConstIterator::~ConstIterator(), DLList::~DLList(), and Iterator::~Iterator().

```
00135 {
00136         if (--references<=0)
00137         {
00138                 delete this;
00139                 return 1;
00140         }
00141
```

4.56.2.2 long RefRep::Reference () [inline]

Definition at line 120 of file refrep.h.

Referenced by AMatrix::AMatrix(), ConstIterator::ConstIterator(), ConstIterator::Copy(), Iterator::Copy(), DLList::DLList(), Iterator::Iterator(), DLList::operator=(), and AMatrix::operator=().

```
00121 { return ++references; }
00122
```

4.56.2.3 long RefRep::References () const [inline]

Definition at line 148 of file refrep.h.

```
00149 { return references; }
00150
```

4.56.3 Member Data Documentation

4.56.3.1 long RefRep::references [private]

Definition at line 78 of file refrep.h.

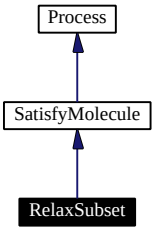
The documentation for this class was generated from the following file:

- **refrep.h**

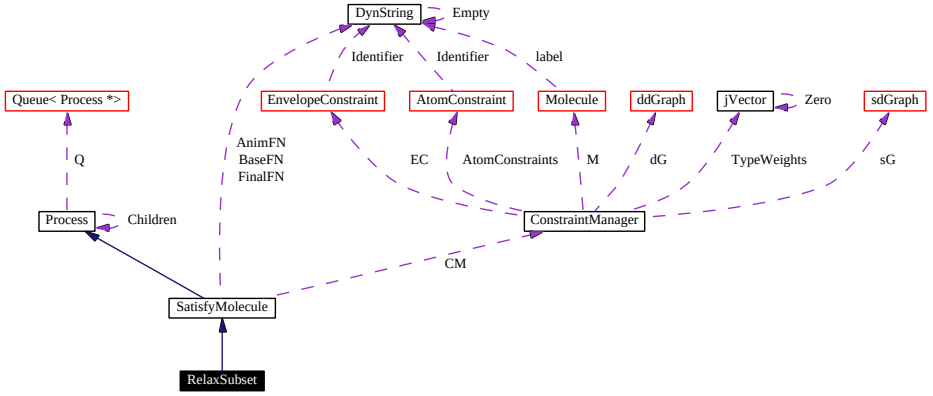
4.57 RelaxSubset Class Reference

`#include <jd_relax_subset.h>`

Inheritance diagram for RelaxSubset



Collaboration diagram for RelaxSubset:



Public Methods

- **RelaxSubset** (const RelaxSubset &ca)
- **RelaxSubset** (**ConstraintManager** &**CM**, double in_fraction=0.5)
- **~RelaxSubset** ()
- int **run** ()
- void **setFraction** (double F)

  set the fraction of worst atoms {0..1} to be affected.

Private Attributes

- int **NumToRelax**

  == (int) (number of atoms)(fraction to relax).*

- double* **PriorityList**
- double **Fraction**

  Fraction' defines the number of worst atoms {0..1} to be affected.

4.57.1 Detailed Description

//

File: **jd_relax_subset.h**

This is a 'job' that works in the GRAPE system. It relaxes a subset of the molecule's atoms.

//

Definition at line 21 of file jd_relax_subset.h.

4.57.2 Constructor & Destructor Documentation

4.57.2.1 RelaxSubset::RelaxSubset (const RelaxSubset & *ca*)

Definition at line 3 of file jd_relax_subset.C.

```
00004   : SatisfyMolecule(*(ca.CM))
00005 {
00006   // FIX THIS...  I don't know what goes here.  -jd
00007   DIE("relax subset copt constructor called ");
00008
```

4.57.2.2 RelaxSubset::RelaxSubset (ConstraintManager & *cm_in*, double *Fraction_in* = 0.5)

It is assumed that fraction is between 0 and 1. The behavior of RelaxSubset when passed a double >= 1 will be to call a RelaxAtom job on all of the atoms in the molecule. If the constructor recieves a negative double, nothing is done to the molecule, and a error message is written to Standard Error.

Definition at line 11 of file jd_relax_subset.C.

```
00012   : SatisfyMolecule(cm_in)
00013   {
00014
00015     Fraction = Fraction_in;
00016
00017     // Create one child process for each atom in the list. Later only
00018     // the 'fraction' of these atoms will be queued to run.
00019     numC = CM->numAtoms();
00020
00021     Children = new Process*[numC];
00022
00023     PriorityList = new double [numC];
00024
00025     // Create a new relaxation process for every atom in the molecule:
00026     for(int i=0; i<numC; ++i)
00027       Children[i] = new gwRelaxAtom(*CM,i);
00028
```

4.57.2.3 RelaxSubset::~RelaxSubset ()

Definition at line 30 of file jd_relax_subset.C.

```
00031   {
```

```
00032     // Don't delete my children (since Process takes care of it).
00033     // i.e. dont do for(int i=0; i<numChildren; ++i) delete Children[i].
00034     // or delete[] Children.
00035
00036     //    CM_ptr=0;
00037     delete [] PriorityList;
00038
```

4.57.3 Member Function Documentation

4.57.3.1 int RelaxSubset::run (void) [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 40 of file jd_relax_subset.C.

```
00041   {
00042 #ifdef JOB_VERBOSE
00043     cerr << "Start: \tRelaxSubset::run()"<<endl;
00044 #endif
00045
00046     // DEBUG
00047     if (Fraction < 0 || Fraction > 1) {
00048       cerr << "tried to call RelaxSubset on a fraction = " << Fraction << endl;
00049       return (-1);
00050     }
00051
00052     NumToRelax = (int) (Fraction * numC );
00053
00054     // Compute the priorities of the children, based on their
00055     // residuals
00056     for(int i=0; i<numC; ++i)
00057       {
00058         double tempPriority = CM->stress(i);
00059
00060         Children[i]->setPriority(tempPriority);
00061         PriorityList[i] = tempPriority;
00062
00063 #ifdef JOB_VERBOSE3
00064         cerr << "Set:   \t" << "Priority for atom " <<i<<"="<< tempPriority << endl;
00065 #endif
00066       }
00067
00068     //double Cutoff = LinearSelect(PriorityList, NumToRelax, numC);
00069
00070     // Queue children to run queue
00071     for(int i=0; i<numC; ++i)
00072       {
00073         //if (Children[i]->getPriority() > Cutoff) addChild(Children[i]);
00074         addChild(Children[i]); // add em all
00075       }
00076
00077     // now do the queue!
00078     for(int i=0; i<NumToRelax; ++i){
00079       runNextChild();
00080     }
00081
```

4.57.3.2 void RelaxSubset::setFraction (double *F*) [inline]

set the fraction of worst atoms {0..1} to be affected.
Definition at line 40 of file jd_relax_subset.h.

```
00040                           {Fraction = F;}
00041
```

4.57.4 Member Data Documentation

4.57.4.1 double RelaxSubset::Fraction [private]

Fraction' defines the number of worst atoms {0..1} to be affected.

Definition at line 52 of file jd_relax_subset.h.

4.57.4.2 int RelaxSubset::NumToRelax [private]

== (int) (number of atoms)∗(fraction to relax).

Definition at line 48 of file jd_relax_subset.h.

4.57.4.3 double ∗ RelaxSubset::PriorityList [private]

Definition at line 49 of file jd_relax_subset.h.

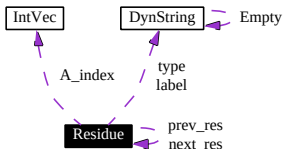
The documentation for this class was generated from the following files:

- **jd_relax_subset.h**
- **jd_relax_subset.C**

4.58 Residue Class Reference

`#include <residue.h>`

Collaboration diagram for Residue:



Public Methods

- const int& **ID** () const

  need residue_registry need to point to atoms need to have atoms point to residue.

- int& **ID** ()
- const **DynString**& **Type** () const
- **DynString**& **Type** ()
- const **DynString**& **Label** () const
- **DynString**& **Label** ()
- const **jVector**& **operator** * () const
- **jVector**& **operator** * ()
- int **Setup** (**Molecule** &Mol, int start_atom, Residue *prev_res_in, Residue *next_res_in)
- int **Compare** (const Residue &a) const
- const **IntVec**& **GetAtomList** () const
- const Residue& **operator=** (const Residue &res)
- int **Output** (ostream &os) const
- int **Input** (istream &is)
- int **OutputBinary** (ostream &os) const
- int **InputBinary** (istream &is)
- void **Copy** (const Residue &res)
- **MAKECOMPARISONS_WITH_COMPARE** (Residue)
- **MAKE_IO_STREAM_OPERATORS** (Residue)
- **MAKE_BIN_STREAM_OPERATORS** (Residue)

Private Methods

- **Residue** (int identifier=UNINITIALIZED_RES_ID)
- void **AddAtom** (**Molecule** &Mol, int atom_id, const **DynString** &atom_type, const char *label_end)
- void **AddBond** (**Molecule** &Mol, int atom_id1, int atom_id2)

Private Attributes

- int **id**
- int **Type_index**
- **DynString label**

- **DynString type**
- int **numAtoms**
- int **numBonds**
- **IntVec A_index**
- int **N_chain_atom**
- int **CA_chain_atom**
- int **O_chain_atom**
- int **C_chain_atom**
- int **CB_chain_atom**
- Residue* **next_res**
- Residue* **prev_res**

Friends

- class **Molecule**
- class **ResidueRegistry**

4.58.1 Constructor & Destructor Documentation

4.58.1.1 Residue::Residue (int *identifier* = UNINITIALIZED_RES_ID) [inline, private]

Definition at line 119 of file residue.h.

```
00119                                    {
00120   id=identifier;
00121   type.Clear();
00122   Type_index = UNINITIALIZED_RES_ID;
00123   label = "uninitialized residue";
00124   numAtoms = -1;
00125   numBonds = -1;
00126   N_chain_atom = UNINITIALIZED_ATOM_ID;
00127   CA_chain_atom = UNINITIALIZED_ATOM_ID;
00128   O_chain_atom = UNINITIALIZED_ATOM_ID;
00129   C_chain_atom = UNINITIALIZED_ATOM_ID;
00130   CB_chain_atom = UNINITIALIZED_ATOM_ID;
00131   next_res = 0;
00132   prev_res = 0;
00133
00134
```

4.58.2 Member Function Documentation

4.58.2.1 void Residue::AddAtom (Molecule & *Mol*, int *atom_id*, const DynString & *atom_type*, const char * *label_end*) [private]

Definition at line 13 of file residue.C.

Referenced by Setup().

```
00013                                                                                {
00014
00015
00016   DynString label = "A:";
00017   label += atom_id;
```

```
00018   label += " ";
00019   label += ResidueRegistry::code_3[Type_index];
00020   label += id;
00021   label += " ";
00022   label += label_end;
00023
00024   //cerr << "residue.C:: Adding atom:"<< atom_id << " as " <<  label << endl;
00025
00026   A_index.Append(atom_id);
00027   AtomRegistry::SetType (Mol.A[atom_id],atom_type, label, id);
00028
00029
```

4.58.2.2 void Residue::AddBond (Molecule & *Mol*, int *atom_id1*, int *atom_id2*) [private]

Definition at line 5 of file residue.C.

Referenced by Setup().

```
00005                                                                {
00006   //    cerr << "bonding " <<  Mol[atom_id1].Label() << "  TO  " << Mol[atom_id2].Label() <<endl;
00007   Mol[atom_id1].AddBond(atom_id2);
00008   Mol[atom_id2].AddBond(atom_id1);
00009
00010
```

4.58.2.3 int Residue::Compare (const Residue & *a*) const [inline]

Definition at line 154 of file residue.h.

```
00155 { return (id-a.id); }
00156
```

4.58.2.4 void Residue::Copy (const Residue & *res*)

Definition at line 399 of file residue.C.

Referenced by operator=().

```
00399                                       {
00400
00401   id = res.id;
00402   type = res.type;
00403   Type_index = res.Type_index;
00404   label = res.label;
00405   numAtoms = res.numAtoms;
00406   numBonds = res.numBonds;
00407   A_index = res.A_index;
00408   N_chain_atom = res.N_chain_atom;
00409   CA_chain_atom = res.CA_chain_atom;
00410   O_chain_atom = res.O_chain_atom;
00411   C_chain_atom = res.C_chain_atom;
00412   CB_chain_atom = res.CB_chain_atom;
00413   next_res = res.next_res;
00414   prev_res = res.prev_res;
00415
00416
```

4.58.2.5 const IntVec & Residue::GetAtomList () const [inline]

Definition at line 136 of file residue.h.

```
00137 {  return A_index; }
00138
```

4.58.2.6 int & Residue::ID () [inline]

Definition at line 141 of file residue.h.

```
00142 { return id; }
00143
```

4.58.2.7 const int & Residue::ID () const [inline]

need residue_registry need to point to atoms need to have atoms point to residue.

Definition at line 139 of file residue.h.
Referenced by ResidueRegistry::SetType_code_1(), and Molecule::readFasta().

```
00140 { return id; }
00141
```

4.58.2.8 int Residue::Input (istream & *is*) [inline]

Definition at line 161 of file residue.h.

```
00162 { is >> id ; return id; }
00163
```

4.58.2.9 int Residue::InputBinary (istream & *is*) [inline]

Definition at line 168 of file residue.h.

```
00169     { is >= id; is >= pos; return id; }
00170
```

4.58.2.10 DynString & Residue::Label () [inline]

Definition at line 151 of file residue.h.

```
00152 { return label; }
00153
```

4.58.2.11 const DynString & Residue::Label () const [inline]

Definition at line 149 of file residue.h.
Referenced by ResidueRegistry::SetType_code_1().

```
00150 { return label; }
00151
```

4.58.2.12 Residue::MAKECOMPARISONS_WITH_COMPARE (Residue)

4.58.2.13 Residue::MAKE_BIN_STREAM_OPERATORS (Residue)

4.58.2.14 Residue::MAKE_IO_STREAM_OPERATORS (Residue)

4.58.2.15 int Residue::Output (ostream & *os*) const [inline]

Definition at line 157 of file residue.h.

```
00158     { os << id << ' ' << pos; return id; }
00159
```

4.58.2.16 int Residue::OutputBinary (ostream & *os*) const [inline]

Definition at line 164 of file residue.h.

```
00165     { os <= id; os <= pos; return id; }
00166
```

4.58.2.17 int Residue::Setup (Molecule & *Mol*, int *start_atom*, Residue * *prev_res_in*, Residue * *next_res_in*)

sets up appropraite atoms in **Molecule** (p. 356) adds bond information within atoms returns last atom handled+1 (the next unhandled atom), UNINITIALIZED_ATOM_ID on err

Definition at line 31 of file residue.C.

Referenced by Molecule::readFasta().

```
00031                                                                                {
00032   A_index.Resize(0);
00033
00034   next_res = next_res_in;
00035   prev_res = prev_res_in;
00036
00037   DynString N_type_atom('N');
00038   DynString C_type_atom('C');
00039   DynString O_type_atom('O');
00040   DynString S_type_atom('S');
00041
00042
00043   if (Type_index >= 20) {
00044     AddAtom(Mol, start_atom+0, C_type_atom, "Dummy Carbon");
00045     N_chain_atom = CA_chain_atom = O_chain_atom = CA_chain_atom =
00046       CB_chain_atom = start_atom+0;
00047
00048     // connect residues
00049     if (prev_res && prev_res->C_chain_atom >= 0)
00050       AddBond(Mol, prev_res->C_chain_atom, start_atom);
00051
00052     // doing both to ensure connection in non ordered entry case
00053     if (next_res && next_res->N_chain_atom >= 0)
00054       AddBond(Mol, start_atom+2, next_res->N_chain_atom);
00055
00056   } else {
00057
00058     // add the protein chain atoms
```

```
00059     AddAtom(Mol, start_atom+0, N_type_atom, "N_chain");
00060     N_chain_atom = start_atom+0;
00061
00062     AddAtom(Mol, start_atom+1, C_type_atom, "CA_chain");
00063     CA_chain_atom = start_atom+1;
00064
00065     AddAtom(Mol, start_atom+2, C_type_atom, "C_chain");
00066     C_chain_atom = start_atom+2;
00067
00068     AddAtom(Mol, start_atom+3, O_type_atom, "O_chain");
00069     O_chain_atom = start_atom+3;
00070
00071     CB_chain_atom = start_atom+4; // reset for Glycine
00072
00073
00074     // connect residues
00075     if (prev_res && prev_res->C_chain_atom >= 0)
00076       AddBond(Mol, prev_res->C_chain_atom, start_atom); // C(prev) -> N
00077
00078     // add the standard chain bonds
00079     AddBond(Mol, start_atom, start_atom+1); // N -> CA
00080     AddBond(Mol, start_atom+1, start_atom+2); // CA -> C
00081     AddBond(Mol, start_atom+2, start_atom+3); // C -> O
00082
00083     // doing both to ensure connection in non ordered entry case
00084     if (next_res && next_res->N_chain_atom >= 0)
00085       AddBond(Mol, start_atom+2, next_res->N_chain_atom); // C -> N(next)
00086
00087   }
00088
00089   // switch adds stoms and bonds approrpiate for each residue type
00090   // & fixes CB in glycine
00091
00092   switch (Type_index){ // here we encode residue differences
00093
00094     //         0   1   2   3   4   5   6   7   8
00095     //       " N   CA  C   O   CB ", /* ala */
00096     //       " N   CA  C   O   CB  SG ", /* cys */
00097     //       " N   CA  C   O   CB  CG  OD1 OD2", /* asp */
00098     //       " N   CA  C   O   CB  CG  CD  OE1 OE2", /* glu */
00099     //       " N   CA  C   O   CB  CG  CD1 CD2 CE1 CE2 CZ ", /* phe */
00100     //       " N   CA  C   O   C  ", /* gly */
00101
00102     // 0   A  alanine
00103   case 0:
00104     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00105     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00106     break;
00107
00108     // 1   C  cystine
00109   case 1:
00110     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00111     AddAtom(Mol, start_atom+5, S_type_atom, "SG");
00112     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00113     AddBond(Mol, start_atom+5, CB_chain_atom);
00114     break;
00115
00116     // 2   D  aspartate asp
00117   case 2:
00118     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00119     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00120     AddAtom(Mol, start_atom+6, O_type_atom, "OD1");  // O_delta1 atom
00121     AddAtom(Mol, start_atom+7, O_type_atom, "OD2");  // O_delta2 atom
00122     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00123     AddBond(Mol, CB_chain_atom, start_atom+5);
```

```
00124     AddBond(Mol, start_atom+5, start_atom+6);
00125     AddBond(Mol, start_atom+5, start_atom+7);
00126
00127     break;
00128     // 3   E  glutamate
00129   case 3:
00130     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00131     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00132     AddAtom(Mol, start_atom+6, C_type_atom, "CD");  // C_delta atom
00133     AddAtom(Mol, start_atom+7, O_type_atom, "OE1");  // O_epsilon1 atom
00134     AddAtom(Mol, start_atom+8, O_type_atom, "OE2");  // O_epsilon2 atom
00135     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00136     AddBond(Mol, CB_chain_atom, start_atom+5);
00137     AddBond(Mol, start_atom+5, start_atom+6);
00138     AddBond(Mol, start_atom+6, start_atom+7);
00139     AddBond(Mol, start_atom+6, start_atom+8);
00140
00141     break;
00142     // 4   F  phenylalanine
00143   case 4:
00144     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00145     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00146     AddAtom(Mol, start_atom+6, C_type_atom, "CD1");  // C_delta atom
00147     AddAtom(Mol, start_atom+7, C_type_atom, "CD2");  // C_delta atom
00148     AddAtom(Mol, start_atom+8, C_type_atom, "CE1");  // C_epsilon atom
00149     AddAtom(Mol, start_atom+9, C_type_atom, "CE2");  // C_epsilon atom
00150     AddAtom(Mol, start_atom+10, C_type_atom, "CZ");  // C_zeta atom
00151     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00152     AddBond(Mol, CB_chain_atom, start_atom+5);
00153     AddBond(Mol, start_atom+5, start_atom+6);
00154     AddBond(Mol, start_atom+5, start_atom+7);
00155     AddBond(Mol, start_atom+6, start_atom+8);
00156     AddBond(Mol, start_atom+7, start_atom+9);
00157     AddBond(Mol, start_atom+8, start_atom+10);
00158     AddBond(Mol, start_atom+9, start_atom+10);
00159
00160     break;
00161     // 5   G  glycine
00162   case 5:
00163     CB_chain_atom = CA_chain_atom; // not perfect, but close.
00164
00165     break;
00166
00167     //         0   1   2   3   4   5   6   7   8   9
00168     //       " N   CA  C   O   CB  CG  ND1 CD2 CE1 NE2", /* his */
00169     //       " N   CA  C   O   CB  CG1 CG2 CD1", /* ile */
00170     //       " N   CA  C   O   CB  CG  CD  CE  NZ ", /* lys */
00171     //       " N   CA  C   O   CB  CG  CD1 CD2", /* leu */
00172
00173     // 6   H  histidine
00174   case 6:
00175     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00176     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00177     AddAtom(Mol, start_atom+6, N_type_atom, "ND1");  // N_delta atom
00178     AddAtom(Mol, start_atom+7, C_type_atom, "CD2");  // C_delta atom
00179     AddAtom(Mol, start_atom+8, C_type_atom, "CE1");  // C_epsilon atom
00180     AddAtom(Mol, start_atom+9, N_type_atom, "NE2");  // N_epsilon atom
00181     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00182     AddBond(Mol, CB_chain_atom, start_atom+5);
00183     AddBond(Mol, start_atom+5, start_atom+6);
00184     AddBond(Mol, start_atom+5, start_atom+7);
00185     AddBond(Mol, start_atom+6, start_atom+8);
00186     AddBond(Mol, start_atom+7, start_atom+9);
00187     AddBond(Mol, start_atom+8, start_atom+9);
00188     break;
```

```
00189
00190       // 7   I  isoleucine
00191   case 7:
00192     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00193     AddAtom(Mol, start_atom+5, C_type_atom, "CG1");  // C_gamma atom
00194     AddAtom(Mol, start_atom+6, C_type_atom, "CG2");  // C_gamma atom
00195     AddAtom(Mol, start_atom+7, C_type_atom, "CD1");  // C_delta atom
00196     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00197     AddBond(Mol, CB_chain_atom, start_atom+5);
00198     AddBond(Mol, CB_chain_atom, start_atom+6);
00199     AddBond(Mol, start_atom+5, start_atom+7);
00200     break;
00201
00202       // 8   K  lysine
00203   case 8:
00204     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00205     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00206     AddAtom(Mol, start_atom+6, C_type_atom, "CD");  // C_delta atom
00207     AddAtom(Mol, start_atom+7, C_type_atom, "CE");  // C_epsilon atom
00208     AddAtom(Mol, start_atom+8, N_type_atom, "NZ");  // N_zeta atom
00209     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00210     AddBond(Mol, CB_chain_atom, start_atom+5);
00211     AddBond(Mol, start_atom+5, start_atom+6);
00212     AddBond(Mol, start_atom+6, start_atom+7);
00213     AddBond(Mol, start_atom+7, start_atom+8);
00214     break;
00215
00216       // 9   L  leucine
00217   case 9:
00218     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00219     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00220     AddAtom(Mol, start_atom+6, C_type_atom, "CD1");  // C_delta1 atom
00221     AddAtom(Mol, start_atom+7, C_type_atom, "CD2");  // C_delta2 atom
00222     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00223     AddBond(Mol, CB_chain_atom, start_atom+5);
00224     AddBond(Mol, start_atom+5, start_atom+6);
00225     AddBond(Mol, start_atom+5, start_atom+7);
00226     break;
00227
00228     //       " N   CA  C   O   CB  CG  SD  CE ", /* met */
00229     //       " N   CA  C   O   CB  CG  OD1 ND2", /* asn */
00230       // 10  M  methionine
00231   case 10:
00232     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00233     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00234     AddAtom(Mol, start_atom+6, S_type_atom, "SD");  // sulphur
00235     AddAtom(Mol, start_atom+7, C_type_atom, "CE");  // C_epsilon atom
00236     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00237     AddBond(Mol, CB_chain_atom, start_atom+5);
00238     AddBond(Mol, start_atom+5, start_atom+6);
00239     AddBond(Mol, start_atom+6, start_atom+7);
00240     break;
00241
00242       // 11  N  asparagine  asn
00243   case 11:
00244     AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00245     AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00246     AddAtom(Mol, start_atom+6, O_type_atom, "OD1");  // O_delta1 atom
00247     AddAtom(Mol, start_atom+7, N_type_atom, "ND2");  // N_delta2 atom
00248     AddBond(Mol, CA_chain_atom, CB_chain_atom);
00249     AddBond(Mol, CB_chain_atom, start_atom+5);
00250     AddBond(Mol, start_atom+5, start_atom+6);
00251     AddBond(Mol, start_atom+5, start_atom+7);
00252     break;
00253
```

```
00254       //         0   1   2   3   4   5   6   7   8   9
00255       //       " N   CA  C   O   CB  CG  CD ", /* pro */
00256       //       " N   CA  C   O   CB  CG  CD  OE1 NE2", /* gln */
00257       //       " N   CA  C   O   CB  CG  CD  NE  CZ  NH1 NH2", /* arg */
00258       //       " N   CA  C   O   CB  OG ", /* ser */
00259
00260       //  12  P  proline
00261   case 12:
00262         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00263         AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00264         AddAtom(Mol, start_atom+6, C_type_atom, "CD");  // C_delta atom
00265         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00266         AddBond(Mol, CB_chain_atom, start_atom+5);
00267         AddBond(Mol, start_atom+5, start_atom+6);
00268         AddBond(Mol, start_atom+6, N_chain_atom);
00269         break;
00270
00271       //  13  Q  glutamine
00272   case 13:
00273         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00274         AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00275         AddAtom(Mol, start_atom+6, C_type_atom, "CD");  // C_delta atom
00276         AddAtom(Mol, start_atom+7, O_type_atom, "OE1");  // O_epsilon atom
00277         AddAtom(Mol, start_atom+8, N_type_atom, "NE2");  // N_epsilon atom
00278         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00279         AddBond(Mol, CB_chain_atom, start_atom+5);
00280         AddBond(Mol, start_atom+5, start_atom+6);
00281         AddBond(Mol, start_atom+6, start_atom+7);
00282         AddBond(Mol, start_atom+6, start_atom+8);
00283         break;
00284
00285         //  14  R  arginine
00286   case 14:
00287         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00288         AddAtom(Mol, start_atom+5, C_type_atom, "CG");  // C_gamma atom
00289         AddAtom(Mol, start_atom+6, C_type_atom, "CD");  // C_delta atom
00290         AddAtom(Mol, start_atom+7, N_type_atom, "NE2");  // N_epsilon atom
00291         AddAtom(Mol, start_atom+8, C_type_atom, "CZ");  // C_zeta atom
00292         AddAtom(Mol, start_atom+9, N_type_atom, "NH1");
00293         AddAtom(Mol, start_atom+10, N_type_atom, "NH2");
00294         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00295         AddBond(Mol, CB_chain_atom, start_atom+5);
00296         AddBond(Mol, start_atom+5, start_atom+6);
00297         AddBond(Mol, start_atom+6, start_atom+7);
00298         AddBond(Mol, start_atom+7, start_atom+8);
00299         AddBond(Mol, start_atom+8, start_atom+9);
00300         AddBond(Mol, start_atom+8, start_atom+10);
00301         break;
00302
00303         //   15  S  serine
00304   case 15:
00305         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00306         AddAtom(Mol, start_atom+5, O_type_atom, "OG");  // O_gamma atom
00307         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00308         AddBond(Mol, CB_chain_atom, start_atom+5);
00309         break;
00310
00311       //         0   1   2   3   4   5   6   7   8   9   10  11  12  13
00312       //       " N   CA  C   O   CB  OG1 CG2", /* thr */
00313       //       " N   CA  C   O   CB  CG1 CG2", /* val */
00314       //       " N   CA  C   O   CB  CG  CD1 CD2 NE1 CE2 CE3 CZ2 CZ3 CH2", /* trp */
00315       //       " N   CA  C   O   CB  CG  CD1 CD2 CE1 CE2 CZ  OH ", /* tyr */
00316
00317
00318       //  16  T  threonine
```

```
00319   case 16:
00320         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00321         AddAtom(Mol, start_atom+5, O_type_atom, "OG1");  // O_gamma atom
00322         AddAtom(Mol, start_atom+6, C_type_atom, "CG2");  // C_gamma atom
00323         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00324         AddBond(Mol, CB_chain_atom, start_atom+5);
00325         AddBond(Mol, CB_chain_atom, start_atom+6);
00326         break;
00327
00328       //  17  V  valine
00329   case 17:
00330         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00331         AddAtom(Mol, start_atom+5, C_type_atom, "CG1");  // C_gamma atom
00332         AddAtom(Mol, start_atom+6, C_type_atom, "CG2");  // C_gamma atom
00333         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00334         AddBond(Mol, CB_chain_atom, start_atom+5);
00335         AddBond(Mol, CB_chain_atom, start_atom+6);
00336         break;
00337
00338       //   18  W  tryptophan
00339   case 18:
00340         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00341         AddAtom(Mol, start_atom+5, C_type_atom, "CG");
00342         AddAtom(Mol, start_atom+6, C_type_atom, "CD1");
00343         AddAtom(Mol, start_atom+7, C_type_atom, "CD2");
00344         AddAtom(Mol, start_atom+8, N_type_atom, "NE1");
00345         AddAtom(Mol, start_atom+9, C_type_atom, "CE2");
00346         AddAtom(Mol, start_atom+10, C_type_atom, "CE3");
00347         AddAtom(Mol, start_atom+11, C_type_atom, "CZ2");
00348         AddAtom(Mol, start_atom+12, C_type_atom, "CZ3");
00349         AddAtom(Mol, start_atom+13, C_type_atom, "CH2");
00350
00351         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00352         AddBond(Mol, CB_chain_atom, start_atom+5);
00353         AddBond(Mol, start_atom+5, start_atom+6);
00354         AddBond(Mol, start_atom+5, start_atom+7);
00355         AddBond(Mol, start_atom+6, start_atom+8);
00356         AddBond(Mol, start_atom+8, start_atom+10);
00357         AddBond(Mol, start_atom+7, start_atom+10);
00358         AddBond(Mol, start_atom+7, start_atom+9);
00359         AddBond(Mol, start_atom+10, start_atom+12);
00360         AddBond(Mol, start_atom+9, start_atom+11);
00361         AddBond(Mol, start_atom+11, start_atom+13);
00362         AddBond(Mol, start_atom+12, start_atom+13);
00363
00364         break;
00365       //   19  Y  tyrosine
00366   case 19:
00367         AddAtom(Mol, start_atom+4, C_type_atom, "CB");  // C_beta atom
00368         AddAtom(Mol, start_atom+5, C_type_atom, "CG");
00369         AddAtom(Mol, start_atom+6, C_type_atom, "CD1");
00370         AddAtom(Mol, start_atom+7, C_type_atom, "CD2");
00371         AddAtom(Mol, start_atom+8, C_type_atom, "CE1");
00372         AddAtom(Mol, start_atom+9, C_type_atom, "CE2");
00373         AddAtom(Mol, start_atom+10, C_type_atom, "CZ");
00374         AddAtom(Mol, start_atom+11, O_type_atom, "OH");
00375
00376         AddBond(Mol, CA_chain_atom, CB_chain_atom);
00377         AddBond(Mol, CB_chain_atom, start_atom+5);
00378         AddBond(Mol, start_atom+5, start_atom+6);
00379         AddBond(Mol, start_atom+5, start_atom+7);
00380         AddBond(Mol, start_atom+6, start_atom+8);
00381         AddBond(Mol, start_atom+7, start_atom+9);
00382         AddBond(Mol, start_atom+8, start_atom+10);
00383         AddBond(Mol, start_atom+9, start_atom+10);
```

```
00384         AddBond(Mol, start_atom+10, start_atom+11);
00385
00386       break;
00387       //  20  X  Dummy
00388   case 20:
00389     // do nothing
00390       break;
00391   }
00392
```

4.58.2.18 DynString & Residue::Type () [inline]

Definition at line 146 of file residue.h.

```
00147 { return type; }
00148
```

4.58.2.19 const DynString & Residue::Type () const [inline]

Definition at line 144 of file residue.h.
Referenced by ResidueRegistry::SetType_code_1().

```
00145 { return type; }
00146
```

4.58.2.20 jVector& Residue::operator * ()

4.58.2.21 const jVector& Residue::operator * () const

4.58.2.22 const Residue & Residue::operator= (const Residue & *res*)

Definition at line 396 of file residue.C.

```
00397   { Copy(res); return *this; }
00398
```

4.58.3 Friends And Related Function Documentation

4.58.3.1 class Molecule [friend]

Definition at line 29 of file residue.h.

4.58.3.2 class ResidueRegistry [friend]

Definition at line 30 of file residue.h.

4.58.4 Member Data Documentation

4.58.4.1 IntVec Residue::A_index [private]

Definition at line 102 of file residue.h.

4.58.4.2 int Residue::CA_chain_atom [private]

Definition at line 108 of file residue.h.

4.58.4.3 int Residue::CB_chain_atom [private]

Definition at line 111 of file residue.h.

4.58.4.4 int Residue::C_chain_atom [private]

Definition at line 110 of file residue.h.

4.58.4.5 int Residue::N_chain_atom [private]

Definition at line 107 of file residue.h.

4.58.4.6 int Residue::O_chain_atom [private]

Definition at line 109 of file residue.h.

4.58.4.7 int Residue::Type_index [private]

Definition at line 94 of file residue.h.

4.58.4.8 int Residue::id [private]

Definition at line 91 of file residue.h.

4.58.4.9 DynString Residue::label [private]

Definition at line 96 of file residue.h.

4.58.4.10 Residue * Residue::next_res [private]

Definition at line 113 of file residue.h.

4.58.4.11 int Residue::numAtoms [private]

Definition at line 99 of file residue.h.

4.58.4.12 int Residue::numBonds [private]

Definition at line 100 of file residue.h.

4.58.4.13 Residue * Residue::prev_res [private]

Definition at line 114 of file residue.h.

4.58.4.14 DynString Residue::type [private]

Definition at line 97 of file residue.h.

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

- **residue.h**
- **residue.C**

4.59 ResidueRegistry Class Reference

`#include <residue_registry.h>`

Static Public Methods

- int **CheckType_code_1** (**DynString** &in_label)

  checks if this is a valid 1 code, return -1 or species number (position in arrays below).

- int **SetType_code_1** (**Residue** &r, **DynString** &in_label)

  sets reside to label type.

Static Public Attributes

- const char* **names** [num_known_residue_species]

  names of the residues.

- const char* **code_1** [num_known_residue_species]

  names of the residues.

- const char* **code_3** [num_known_residue_species]

  names of the residues.

- const int **numAtoms** [num_known_residue_species]

  number of atoms in the residues.

- const int **numBonds** [num_known_residue_species]

  number of atoms in the residues.

4.59.1 Member Function Documentation

4.59.1.1 int ResidueRegistry::CheckType_code_1 (DynString & *in_label*) [inline, static]

checks if this is a valid 1 code, return -1 or species number (position in arrays below).

Definition at line 91 of file residue_registry.h.

Referenced by Molecule::readFasta().

```
00091                                                          {
00092   for (int i = 0; i < num_known_residue_species; i++){
00093     if (in_label == ResidueRegistry::code_1[i])
00094       return i;
00095   }
00096
```

4.59.1.2 int ResidueRegistry::SetType_code_1 (Residue & *r*, DynString & *in_type*) [inline, static]

sets reside to label type.

Definition at line 100 of file residue_registry.h.

Referenced by Molecule::readFasta().

```
00100                                                                    {
00101   for (int i = 0; i < num_known_residue_species; i++){
00102     if (in_type == ResidueRegistry::code_1[i]){
00103       r.Type_index = i;
00104       r.Type() = in_type;
00105       r.Label() = names[i];
00106       r.numAtoms = numAtoms[i];
00107       r.numBonds = numBonds[i];
00108
00109       return r.ID();
00110     }
00111   }
00112
```

4.59.2 Member Data Documentation

4.59.2.1 const char * ResidueRegistry::code_1[num_known_residue_species] [static]

Initial value:

```
 {
  "A", "C", "D", "E", "F", "G", "H", "I", "K", "L", \
  "M", "N", "P", "Q", "R", "S", "T", "V", "W", "Y", \
  "X"
}
```

names of the residues.

Definition at line 174 of file globals.C.

4.59.2.2 const char * ResidueRegistry::code_3[num_known_residue_species] [static]

Initial value:

```
 {
  "ALA", "CYS", "ASP", "GLU", "PHE", "GLY", "HIS", "ILE", "LYS", "LEU", \
  "MET", "ASN", "PRO", "GLN", "ARG", "SER", "THR", "VAL", "TRP", "TYR", \
  "DUM"
}
```

names of the residues.

Definition at line 181 of file globals.C.

4.59.2.3 const char * ResidueRegistry::names[num_known_residue_species] [static]

Initial value:

```
 {
  "alanine","cystine","aspartate","glutamate","phenylalanine","glycine", \
  "histidine","isoleucine","lysine","leucine","methionine", \
  "asparagine","proline","glutamine","arginine","serine","threonine", \
  "valine","tryptophan","tyrosine","dummy"
}
```

names of the residues.

Definition at line 166 of file globals.C.

4.59.2.4 const int ResidueRegistry::numAtoms[num_known_residue_species] [static]

Initial value:

```
 {
  5,6,8,9,11,4,10,8,9,8,8,8,7,9,11,6,7,7,14,12,  1
}
```

number of atoms in the residues.

Definition at line 188 of file globals.C.

4.59.2.5 const int ResidueRegistry::numBonds[num_known_residue_species] [static]

Initial value:

```
 {

  1,2,4,5, 8,0, 7,4,5,4,4,4,4,5, 7,2,3,3,12, 9,  0

}
```

number of atoms in the residues.

Definition at line 213 of file globals.C.

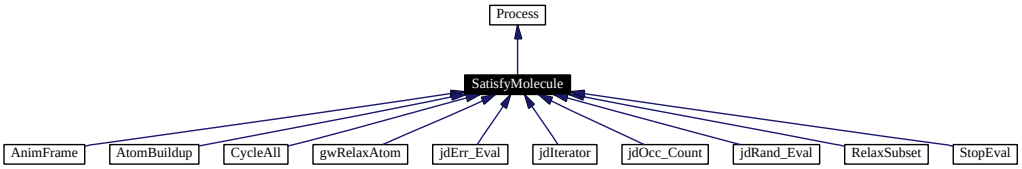
The documentation for this class was generated from the following files:

- **residue_registry.h**
- **globals.C**

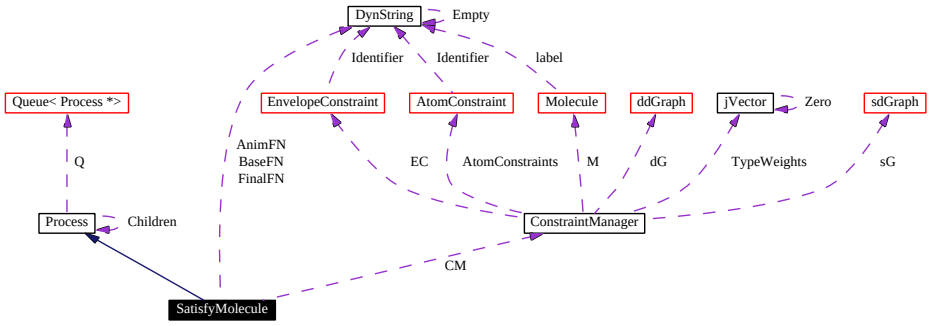
4.60 SatisfyMolecule Class Reference

`#include <satisfy_molecule.h>`

Inheritance diagram for SatisfyMolecule



Collaboration diagram for SatisfyMolecule:



Public Methods

- virtual **~SatisfyMolecule** ()
  destructor.
- void **setMolecule** (const **Molecule** &mol)
- const **Molecule**& **getMolecule** () const

Static Public Attributes

- **DynString BaseFN** = "Base"
- **DynString AnimFN** = "Anim"
- **DynString FinalFN** = "Final"

Protected Methods

- **SatisfyMolecule** (**ConstraintManager** &cm)
  Constructors.
- **SatisfyMolecule** (const SatisfyMolecule &sm)

Protected Attributes

- **ConstraintManager*** **CM**

4.60.1 Detailed Description

//

File: satisfy_molucle.h

This class is a place-holder class. It combines the **ConstraintManager** (p. 97) and the **Process** (p. 391) classes. It is very much like the ManipulateMolecule class except that all accesses to the **Molecule** (p. 356) are done through the **ConstraintManager** (p. 97) that is associated with an instance of the process. This allows the **ConstraintManager** (p. 97) to perform smart updates and fast computations (wrt to satisfy() and dSatisfy() calls).

//

Definition at line 23 of file satisfy_molecule.h.

4.60.2 Constructor & Destructor Documentation

4.60.2.1 SatisfyMolecule::~SatisfyMolecule () [inline, virtual]

destructor.

Definition at line 26 of file satisfy_molecule.h.

```
00026                           { CM = 0; }
00027
```

4.60.2.2 SatisfyMolecule::SatisfyMolecule (ConstraintManager & *cm*) [inline, protected]

Constructors.

Definition at line 47 of file satisfy_molecule.h.

```
00047                                       { CM = &cm; }
00048
```

4.60.2.3 SatisfyMolecule::SatisfyMolecule (const SatisfyMolecule & *sm*) [inline, protected]

Definition at line 48 of file satisfy_molecule.h.

```
00049   { CM = sm.CM; }
00050
```

4.60.3 Member Function Documentation

4.60.3.1 const Molecule & SatisfyMolecule::getMolecule () const [inline]

Definition at line 35 of file satisfy_molecule.h.

```
00036   { return CM->getMolecule(); }
00037
```

4.60.3.2 void SatisfyMolecule::setMolecule (const Molecule & *mol*) [inline]

Definition at line 33 of file satisfy_molecule.h.

```
00034   { CM->setMolecule(mol); }
00035
```

4.60.4 Member Data Documentation

4.60.4.1 DynString SatisfyMolecule::AnimFN = "Anim" [static]

Definition at line 44 of file globals.C.

4.60.4.2 DynString SatisfyMolecule::BaseFN = "Base" [static]

Reimplemented in **jdErr_Eval** (p. 314), **jdOcc_Count** (p. 320), and **jdRand_Eval** (p. 323).

Definition at line 43 of file globals.C.

4.60.4.3 ConstraintManager * SatisfyMolecule::CM [protected]

Definition at line 51 of file satisfy_molecule.h.

4.60.4.4 DynString SatisfyMolecule::FinalFN = "Final" [static]

Definition at line 45 of file globals.C.

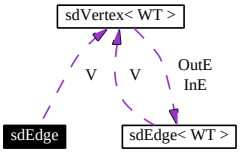
The documentation for this class was generated from the following files:

- **satisfy_molecule.h**
- **globals.C**

4.61 sdEdge Class Template Reference

`#include <sdvertex.h>`

Collaboration diagram for sdEdge:



Public Methods

- **sdEdge** ()
- **sdEdge** (const sdEdge< WT > &e)
- **~sdEdge** ()
- int **Compare** (const sdEdge< WT > &e) const
- **MAKECOMPARISONS_WITH_COMPARE** (sdEdge< WT >)

Public Attributes

- **sdVertex**<WT>* **V**
- WT **W**

Friends

- class **sdVertex< WT >**

template<class WT = double> class sdEdge

4.61.1 Constructor & Destructor Documentation

4.61.1.1 template<class WT = double> sdEdge<WT>::sdEdge<WT> () [inline]

Definition at line 31 of file sdvertex.h.

```
00031                       { V=0; }
00032
```

4.61.1.2 template<class WT = double> sdEdge<WT>::sdEdge<WT> (const sdEdge< WT > & *e*) [inline]

Definition at line 32 of file sdvertex.h.

```
00032                                       { V=e.V; W = e.W; }
00033
```

4.61.1.3 template<class WT = double> sdEdge<WT>::~sdEdge<WT> () [inline]

Definition at line 33 of file sdvertex.h.

```
00033                           { /* empty */ }
00034
```

4.61.2 Member Function Documentation

4.61.2.1 template<class WT> int sdEdge< WT >::Compare (const sdEdge< WT > & *e*) const [inline]

Definition at line 15 of file sdvertex.inc.

```
00016   { return (V->ID - e.V->ID); }
00017
```

4.61.2.2 template<class WT = double> sdEdge<WT>::MAKECOMPARISONS_WITH_COMPARE (sdEdge< WT >)

4.61.3 Friends And Related Function Documentation

4.61.3.1 template<class WT = double> class sdVertex [friend]

Definition at line 29 of file sdvertex.h.

4.61.4 Member Data Documentation

4.61.4.1 template<class WT = double> sdVertex< WT > * sdEdge<WT>::V

Definition at line 38 of file sdvertex.h.

4.61.4.2 template<class WT = double> WT sdEdge<WT>::W

Definition at line 40 of file sdvertex.h.

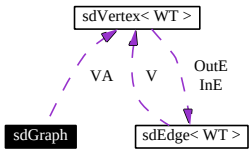
The documentation for this class was generated from the following files:

- **sdvertex.h**
- **sdvertex.inc**

4.62 sdGraph Class Template Reference

`#include <sdgraph.h>`

Collaboration diagram for sdGraph:



Public Methods

- **sdGraph** ()
- **sdGraph** (const sdGraph< DT, WT > &sdg)
- **sdGraph** (**ddGraph**< DT, WT > &ddg, bool KeepIds=false)
- **~sdGraph** ()
- int **NumVertices** () const
- int **NumEdges** () const
- void **Copy** (const sdGraph< DT, WT > &sdg)
- void **Copy** (**ddGraph**< DT, WT > &ddg, bool KeepIds=false)
- const sdGraph<DT,WT>& **operator=** (const sdGraph< DT, WT > &sdg)
- const sdGraph<DT,WT>& **operator=** (const **ddGraph**< DT, WT > &ddg)
- int **GetVertexIndex** (const DT &x) const
- int **HasVertex** (const DT &x) const
- int **HasEdge** (int i, int j) const
- int **OutDegree** (int i) const
- int **InDegree** (int i) const
- int **Degree** (int i) const
- const **sdVertex**<WT>& **GetVertex** (int i) const
- **sdVertex**<WT>& **GetVertex** (int i)
- const **sdVertex**<WT>& **operator**[] (int i) const
- **sdVertex**<WT>& **operator**[] (int i)
- const DT& **GetData** (int i) const
- DT& **GetData** (int i)
- const DT& **operator()** (int i) const
- DT& **operator()** (int i)
- const WT& **GetWeight** (int i, int j, int alt=0) const
- WT& **GetWeight** (int i, int j, int alt=0)
- const WT& **operator()** (int i, int j, int alt=0) const
- WT& **operator()** (int i, int j, int alt=0)
- sdGraph<DT,WT> **SubGraph** (const **SDG_SetInt** &vertset) const
- int **Output** (ostream &os) const
- int **Input** (istream &is)
- ostream& **PrintDotty** (ostream &os, int edges=0) const
- **KAdjMat AdjMatrix** () const
- **KAdjMat*** **TravMatrices** (int src, int k, **KAdjMat** *P=0) const
- **SDG_IntList**** **TravLists** (int src, int k, **KAdjMat** *P=0) const
- int **InputBinary** (istream &is)

- int **OutputBinary** (ostream &os) const
- **neighbor_iterator begin_neighbor** (int vndx=0, int rev=0)
- **df_iterator begin_df** (int vndx=0, int rev=0)
- **bf_iterator begin_bf** (int vndx=0, int rev=0)
- **path_iterator begin_path** (int src, int tgt, int len, **DLList**< int > *const *tl=0, int(*filter)(const sdGraph< DT, WT > &graph, const int *path, int ndx, int pathlen, void *arbitrary_criteria)=0, void *criteria=0)
- **jVector Dijkstra** (int vndx=0, int rev=0, int wc=0, int *pred=0) const
- int **DFTimeStamps** (int *discover, int *finish, int rev=0) const
- **SDG_SetSetInt SCC** () const
- **SDG_SetSetInt StronglyConnectedComponents** () const
- sdGraph<DT,WT> **SCCGraph** () const
- int **NumSCC** () const

Private Methods

- void **Clear** ()
- int **ValidVertex** (int i) const
- int **ValidData** (int i) const
- int **OutDegree** (const **sdVertex**< WT > *vert) const
- int **InDegree** (const **sdVertex**< WT > *vert) const
- int **Degree** (const **sdVertex**< WT > *vert) const
- void **DF_Visit** (int u, **sdg_color** *color, int *time, int *discover, int *finish, **SDG_SetInt** *treemembers, int rev) const
- int **Initialize_Single_Source** (int vndx, **jVector** &D, int *P) const
- void **Relax** (int u, int v, **jVector** &D, int *P, int rev, int wc) const

Private Attributes

- int **NumV**
- int **NumE**
- **sdVertex**<WT>** **VA**
- DT* **Data**

Friends

- class **ddGraph**< **DT, WT** >
- istream& **operator**>= (istream &is, sdGraph< DT, WT > &sdg)
- ostream& **operator** (ostream &os, const sdGraph< DT, WT > &sdg)

template<class DT, class WT = double> class sdGraph

4.62.1 Constructor & Destructor Documentation

4.62.1.1 template<class DT, class WT> sdGraph< DT, WT >::sdGraph<DT, WT> () [inline]

Definition at line 308 of file sdgraph.inc.

```
00309   {
00310     NumV = 0;
00311     NumE = 0;
00312     VA   = 0;
00313     Data = 0;
00314
```

4.62.1.2 template<class DT, class WT> sdGraph< DT, WT >::sdGraph<DT, WT> (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 317 of file sdgraph.inc.

```
00318   {
00319     NumV = 0;
00320     NumE = 0;
00321     VA = 0;
00322     Data = 0;
00323     Copy(sdg);
00324
```

4.62.1.3 template<class DT, class WT> sdGraph< DT, WT >::sdGraph<DT, WT> (ddGraph< DT, WT > & *ddg*, bool *KeepIds* = false) [inline]

Definition at line 328 of file sdgraph.inc.

```
00329   {
00330     NumV = 0;
00331     NumE = 0;
00332     VA = 0;
00333     Data = 0;
00334     Copy(ddg, KeepIds);
00335
```

4.62.1.4 template<class DT, class WT> sdGraph< DT, WT >::~sdGraph<DT, WT> () [inline]

Definition at line 338 of file sdgraph.inc.

```
00339   { Clear(); }
00340
```

4.62.2 Member Function Documentation

4.62.2.1 template<class DT, class WT> KAdjMat sdGraph< DT, WT >::AdjMatrix () const [inline]

Definition at line 665 of file sdgraph.inc.

Referenced by TravLists(), and TravMatrices().

```
00666   {
00667     KAdjMat A(NumV);
00668     for(int i=0; i<NumV; ++i)
00669         for(int j=0; j<VA[i]->NumO; ++j)
00670             A.Connect(i, VA[i]->OutE[j].V->ID);
00671     return A;
00672
```

4.62.2.2 template<class DT, class WT> void sdGraph< DT, WT >::Clear () [inline, private]

Definition at line 14 of file sdgraph.inc.

Referenced by Copy(), InputBinary(), and ~sdGraph().

```
00015   {
00016     for(int i=0; i<NumV; ++i)
00017         delete VA[i];
00018     if(VA) { delete[] VA; VA = 0; }
00019     if(Data) { delete[] Data; Data = 0; }
00020
```

4.62.2.3 template<class DT, class WT> void sdGraph< DT, WT >::Copy (ddGraph< DT, WT > & *ddg*, bool *KeepIds* = false) [inline]

Definition at line 68 of file sdgraph.inc.

```
00069   {
00070     int numout, numin, ndx, current, neighbor;
00071     map< int, int > IdMap;
00072     map< DT, int > DataMap;
00073     sdVertex<WT> ** VA_tmp=0;
00074
00075     map<DT, ddVertex<WT>* >::const_iterator dynvert_it;
00076     set<ddEdge<WT> >::const_iterator eit;
00077     ddVertex<WT> * vert=0;
00078     ddEdge<WT> edge;
00079
00080     // Delete everything everything:
00081     //
00082     Clear();
00083
00084     if(ddg.NumV<1) return;
00085
00086     // Try to allocate space to hold all of the vertices and the data:
00087     //
00088     if(!(VA = new sdVertex<WT>*[ddg.NumV])) return;
00089
00090     if(!(Data = new DT[ddg.NumV]))
00091       { delete[] VA; VA = 0; return; }
00092
00093     // Temporary vertex storage:
00094     //
00095     if(!(VA_tmp = new sdVertex<WT>*[ddg.NumV]))
00096       { delete[] VA; delete[] Data; VA = 0; Data = 0; return; }
00097
00098     NumV = ddg.NumV;
00099
00100     // Initialize the vertices:
00101     //
00102     for(int i=0; i<NumV; ++i) VA[i] = 0;
00103
00104     // First, we need to loop through all the vertices in the dynamic graph
00105     // and find out how many edges each of them has (so we can create
00106     // enough space to hold everything.  While we're doing that, we might
00107     // as well copy the data into the static vertices.
00108     //
00109     int v=0;
00110
00111     if (KeepIds){
```

```
00112       int MinVertex = v;  // idgen has no interface for this -- assume zero
00113       int MaxVertex = ddg.IDG.Max();
00114
00115       // next 12 lines are repetitive, and should be integrated w/ above -jd
00116       NumV = MaxVertex+1;
00117
00118       delete [] VA;
00119       VA = new sdVertex<WT>*[NumV];
00120
00121       delete [] Data;
00122       Data = Data = new DT[NumV];
00123
00124       delete [] VA_tmp;
00125       VA_tmp = new sdVertex<WT>*[NumV];
00126
00127       for(int i=0; i<NumV; ++i) VA[i] = 0;
00128
00129       //      for(dynvert_it = ddg.VA.begin(); dynvert_it!=ddg.VA.end(); ++dynvert_it)
00130       for(int VertexNumber = MinVertex; VertexNumber < MaxVertex+1; VertexNumber ++)
00131         {
00132           if(v==NumV)
00133             {
00134               // This is an error condition:  the dynamic graph told us there
00135               // were only NumV vertices but here we are getting the NumV+1'th!
00136               //
00137               break;
00138             }
00139
00140           if (ddg.HasVertex(VertexNumber)) {
00141
00142             //    Data[v] = (*dynvert_it).first;
00143             Data[v] = ddg.GetData(VertexNumber);
00144
00145             //    vert = (*dynvert_it).second;
00146             vert = ddg.GetVertex(Data[v]); //discards const on copy function ?? -jd
00147
00148             // Find out how many outbound and inbound edges this vertex has:
00149             //
00150             numout = vert->OutDegree();
00151             numin  = vert->InDegree();
00152
00153             // Create the vertex:
00154             //
00155             //      VA_tmp[v] = new sdVertex<WT>(vert->ID, numout, numin);
00156             VA_tmp[v] = new sdVertex<WT>(VertexNumber, numout, numin);
00157
00158             // Remember that this data was associated with the v_th vertex:
00159             //
00160             DataMap[Data[v]] = v;
00161
00162             // end of case where vertex VertexNumber' exists in ddg
00163           }         else            {
00164             // case where vertex VertexNumber' does not exist in ddg
00165             Data[v] = 0;
00166             VA_tmp[v] = new sdVertex<WT>(VertexNumber, 0, 0);
00167             DataMap[Data[v]] = -1;
00168           }
00169
00170           v++;
00171         }
00172
00173       // NB sorting routine has been removed (!)
00174       // this may effect other routines on sDgraphs
00175       // if sorting does not occur.   -jd
00176
```

```
00177       for(v=0; v<NumV; ++v)
00178         {
00179           ndx = DataMap[Data[v]];
00180           VA[v] = VA_tmp[ndx];
00181 
00182           // This allows us to resolve where edges point:
00183           //
00184           IdMap[VA[v]->ID] = v;
00185 
00186           // Now set the new id simply to its index in the array:
00187           //
00188           VA[v]->ID = v;
00189         }
00190       delete[] VA_tmp;
00191 
00192 
00193     } else { // default:  do not keep same IDs
00194 
00195       for(dynvert_it = ddg.VA.begin(); dynvert_it!=ddg.VA.end(); ++dynvert_it)
00196         {
00197           if(v==NumV)
00198             {
00199               // This is an error condition:  the dynamic graph told us there
00200               // were only NumV vertices but here we are getting the NumV+1'th!
00201               //
00202               break;
00203             }
00204 
00205           Data[v] = ($\ast$dynvert_it).first;
00206           vert = ($\ast$dynvert_it).second;
00207 
00208           // Find out how many outbound and inbound edges this vertex has:
00209           //
00210           numout = vert->OutDegree();
00211           numin  = vert->InDegree();
00212 
00213           // Create the vertex:
00214           //
00215           VA_tmp[v] = new sdVertex<WT>(vert->ID, numout, numin);
00216 
00217           // Remember that this data was associated with the v_th vertex:
00218           //
00219           DataMap[Data[v]] = v;
00220 
00221           v++;
00222         }
00223 
00224       // Now that all the vertices are created we can sort the data and then
00225       // rearrange the vertices (to match up with their respective data):
00226       //
00227       QuickSort(Data, NumV);
00228       for(v=0; v<NumV; ++v)
00229         {
00230           ndx = DataMap[Data[v]];
00231           VA[v] = VA_tmp[ndx];
00232 
00233           // This allows us to resolve where edges point:
00234           //
00235           IdMap[VA[v]->ID] = v;
00236 
00237           // Now set the new id simply to its index in the array:
00238           //
00239           VA[v]->ID = v;
00240         }
00241       delete[] VA_tmp;
```

```
00242
00243     } // end default case -- do not keep same IDs -jd
00244
00245     // Now copy in all the edges.  This involves iterating through all the
00246     // vertices from the dynamic graph again and copying the edges into
00247     // the appropriate vertex.
00248     //
00249
00250     int e;
00251     for(dynvert_it = ddg.VA.begin(); dynvert_it!=ddg.VA.end(); ++dynvert_it)
00252       {
00253         // Get the vertex pointer and then find out where the corresponding
00254         // vertex is located in the static array:
00255         //
00256         vert = ($\ast$dynvert_it).second;
00257         current = IdMap[vert->ID];
00258
00259         // Iterate through the outbound edges
00260         //
00261         e = 0;
00262         for(eit=vert->OutE.begin(); eit!=vert->OutE.end(); ++eit)
00263           {
00264             // Get the edge and then find out the index (in the static array)
00265             // of the vertex this edge refers to:
00266             //
00267             edge     = *eit;
00268             neighbor = IdMap[edge.V->ID];
00269
00270             // Add the edge to this vertex:
00271             //
00272             VA[current]->OutE[e].V = VA[neighbor];
00273             VA[current]->OutE[e].W = edge.W;
00274             e++;
00275           }
00276
00277         // Iterate through the inbound edges
00278         //
00279         e = 0;
00280         for(eit=vert->InE.begin(); eit!=vert->InE.end(); ++eit)
00281           {
00282             edge     = *eit;
00283             neighbor = IdMap[edge.V->ID];
00284
00285             // Add the to<-from edge to this vertex:
00286             //
00287             VA[current]->InE[e].V = VA[neighbor];
00288             VA[current]->InE[e].W = edge.W;
00289             e++;
00290           }
00291
00292         // Sort the edges:
00293         //
00294         VA[current]->SortEdges();
00295       }
00296
```

4.62.2.4 template<class DT, class WT> void sdGraph< DT, WT >::Copy (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 25 of file sdgraph.inc.

Referenced by operator=(), and sdGraph().

```
00026   {
00027     if(this == &sdg) return;
00028
00029     // Delete everything:
00030     //
00031     Clear();
00032
00033     if(sdg.NumV<1) return;
00034     if(!(VA = new sdVertex<WT>*[sdg.NumV])) return;
00035     if(!(Data = new DT[sdg.NumV]))
00036       { delete[] VA; VA = 0; return; }
00037
00038     // Copy the vertices.
00039     for(int v=0; v<sdg.NumV; ++v)
00040         if(!(VA[v] = new sdVertex<WT>(sdg.VA[v]->ID, sdg.VA[v]->NumO, sdg.VA[v]->NumI)))
00041             { cerr << "sdGraph::Copy\n"; raise(SIGQUIT); }
00042
00043     // Copy the edges.
00044     for(int v=0; v<sdg.NumV; ++v)
00045       {
00046         // Copy the outbound edges in.
00047         for(int e=0; e<sdg.VA[v]->NumO; ++e)
00048           {
00049             VA[v]->OutE[e].V = VA[(sdg.VA[v]->OutE[e].V->ID)];
00050             VA[v]->OutE[e].W = sdg.VA[v]->OutE[e].W;
00051           }
00052
00053         // Copy the inbound edges in.
00054         for(int e=0; e<sdg.VA[v]->NumI; ++e)
00055           {
00056             VA[v]->InE[e].V = VA[(sdg.VA[v]->InE[e].V->ID)];
00057             VA[v]->InE[e].W = sdg.VA[v]->InE[e].W;
00058           }
00059
00060         Data[v] = sdg.Data[v];
00061       }
00062
```

4.62.2.5 template<class DT, class WT> int sdGraph< DT, WT >::DFTimeStamps (int * *discover*, int * *finish*, int *rev* = 0) const [inline]

Definition at line 107 of file sdgalg.inc.

Referenced by SCC().

```
00108   {
00109     if(NumV<1) return 0;
00110
00111     int time = -1;
00112
00113     // Color all the vertices white to mark them as unvisited:
00114     //
00115     sdg_color color[NumV];
00116     for(int u=0; u<NumV; ++u)
00117         color[u] = sdg_white;
00118
00119     for(int u=0; u<NumV; ++u)
00120         if(color[u] == sdg_white)
00121             DF_Visit(u, color, &time, discover, finish, 0, rev);
00122     return 1;
00123
```

4.62.2.6 template<class DT, class WT> void sdGraph< DT, WT >::DF_Visit (int *u*, sdg_color * *color*, int * *time*, int * *discover*, int * *finish*, SDG_SetInt * *treemembers*, int *rev*) const [inline, private]

Definition at line 69 of file sdgalg.inc.

Referenced by DFTimeStamps(), and SCC().

```
00072   {
00073     color[u] = sdg_gray;        // vertex u has just been discovered.
00074     if(treemembers)
00075        treemembers->insert(u);
00076        // treemembers->Add(u);
00077
00078     if(discover && time)
00079         discover[u] = ++($\ast$time);
00080
00081     int v;
00082     // Explore edge u-v:
00083     if(!rev)
00084       {
00085         for(int i=0; i<VA[u]->NumO; ++i)
00086           {
00087             v = VA[u]->OutE[i].V->ID;
00088             if(color[v] == sdg_white)
00089                 DF_Visit(v, color, time, discover, finish, treemembers, rev);
00090           }
00091       }
00092     else
00093       {
00094         for(int i=0; i<VA[u]->NumI; ++i)
00095           {
00096             v = VA[u]->InE[i].V->ID;
00097             if(color[v] == sdg_white)
00098                 DF_Visit(v, color, time, discover, finish, treemembers, rev);
00099           }
00100       }
00101
```

4.62.2.7 template<class DT, class WT = double> int sdGraph<DT, WT>::Degree (const sdVertex< WT > * *vert*) const [private]

4.62.2.8 template<class DT, class WT> int sdGraph< DT, WT >::Degree (int *i*) const [inline]

Definition at line 392 of file sdgraph.inc.

```
00393   {
00394     if(!ValidVertex(i))
00395       { cerr << "sdGraph::Degree\n"; raise(SIGQUIT); }
00396
```

4.62.2.9 template<class DT, class WT> jVector sdGraph< DT, WT >::Dijkstra (int *vndx* = 0, int *rev* = 0, int *wc* = 0, int * *pred* = 0) const [inline]

Definition at line 35 of file sdgalg.inc.

```
00037   {
```

```
00038     IntPriorityQueue<double> Q(NumV,1);
00039     jVector D;
00040     int u_ndx, num_nbrs, u_nbr;
00041     double u_pri;
00042
00043     D.Resize(NumV);
00044
00045     Initialize_Single_Source(vndx,D,pred);
00046
00047     // Copy all the vertices into the queue:
00048     for(int v=0; v<NumV; ++v)
00049         Q.Add(v, D[v] );
00050
00051     while(!Q.IsEmpty())
00052       {
00053         Q.Remove(u_ndx, u_pri);
00054         D[u_ndx] = u_pri;
00055
00056         // Relax all the neighbors of u.
00057         num_nbrs = !rev ? GetVertex(u_ndx).NumO : GetVertex(u_ndx).NumI;
00058         for(int j=0; j<num_nbrs; ++j)
00059           {
00060             u_nbr = !rev ? GetVertex(u_ndx)[j].ID : GetVertex(u_ndx)(j).ID;
00061             Relax(u_ndx, u_nbr, D, pred, rev, wc);
00062             Q.Modify(u_nbr, D[u_nbr]);
00063           }
00064       }
00065
```

4.62.2.10 template<class DT, class WT> DT & sdGraph< DT, WT >::GetData (int *i*) [inline]

Definition at line 432 of file sdgraph.inc.

```
00433   {
00434     if(!ValidData(i))
00435       { cerr << "sdGraph::GetData\n"; cerr << NumV << ',' << i << '\n'; raise(SIGQUIT); }
00436
```

4.62.2.11 template<class DT, class WT> const DT & sdGraph< DT, WT >::GetData (int *i*) const [inline]

Definition at line 424 of file sdgraph.inc.

Referenced by ConstraintManager::getAssocAtoms(), ConstraintManager::getConstraint(), ConstraintManager::getParents(), operator()(), and ConstraintManager::~ConstraintManager().

```
00425   {
00426     if(!ValidData(i))
00427       { cerr << "const sdGraph::GetData\n"; cerr << i << '\n'; raise(SIGQUIT); }
00428
```

4.62.2.12 template<class DT, class WT> sdVertex< WT > & sdGraph< DT, WT >::GetVertex (int *i*) [inline]

Definition at line 408 of file sdgraph.inc.

```
00409   {
00410     if(!ValidVertex(i))
00411       { cerr << i << ',' << NumV << ",sdGraph::GetVertex\n"; raise(SIGQUIT); }
00412
```

4.62.2.13 template<class DT, class WT> const sdVertex< WT > & sdGraph< DT, WT >::GetVertex (int *i*) const [inline]

Definition at line 400 of file sdgraph.inc.

Referenced by operator[]().

```
00401   {
00402     if(!ValidVertex(i))
00403       { cerr << "sdGraph::GetVertex\n"; raise(SIGQUIT); }
00404
```

4.62.2.14 template<class DT, class WT> int sdGraph< DT, WT >::GetVertexIndex (const DT & *x*) const [inline]

Definition at line 350 of file sdgraph.inc.

Referenced by HasVertex().

```
00351   { return BinSearch(Data, NumV, x); }
00352
```

4.62.2.15 template<class DT, class WT> WT & sdGraph< DT, WT >::GetWeight (int *i*, int *j*, int *alt* = 0) [inline]

Definition at line 458 of file sdgraph.inc.

```
00459   {
00460     if(!ValidVertex(i) || !ValidVertex(j))
00461       { cerr << "sdGraph::GetWeight\n"; raise(SIGQUIT); }
00462
```

4.62.2.16 template<class DT, class WT> const WT & sdGraph< DT, WT >::GetWeight (int *i*, int *j*, int *alt* = 0) const [inline]

Definition at line 448 of file sdgraph.inc.

Referenced by Relax(), and operator()().

```
00449   {
00450     if(!ValidVertex(i) || !ValidVertex(j))
00451       { cerr << "sdGraph::GetWeight\n"; raise(SIGQUIT); }
00452
```

4.62.2.17 template<class DT, class WT> int sdGraph< DT, WT >::HasEdge (int *i*, int *j*) const [inline]

Definition at line 358 of file sdgraph.inc.

Referenced by path_iterator::operator++().

```
00359   {
00360     // Don't have the edge if we don't have one of the vertices:
00361     //
00362     if(i<0 || i>=NumV || j<0 || j>=NumV) return 0;
00363
00364     return VA[i]->HasEdgeTo((*VA[j]));
00365
```

4.62.2.18 template<class DT, class WT> int sdGraph< DT, WT >::HasVertex (const DT & *x*) const [inline]

Definition at line 354 of file sdgraph.inc.

Referenced by ConstraintManager::remove().

```
00355   { return (GetVertexIndex(x) != -1); }
00356
```

4.62.2.19 template<class DT, class WT = double> int sdGraph<DT, WT>::InDegree (const sdVertex< WT > * *vert*) const [private]

4.62.2.20 template<class DT, class WT> int sdGraph< DT, WT >::InDegree (int *i*) const [inline]

Definition at line 384 of file sdgraph.inc.

```
00385   {
00386     if(!ValidVertex(i))
00387       { cerr << "sdGraph::InDegree\n"; raise(SIGQUIT); }
00388
```

4.62.2.21 template<class DT, class WT> int sdGraph< DT, WT >::Initialize_Single_Source (int *vndx*, jVector & *D*, int * *P*) const [inline, private]

Definition at line 5 of file sdgalg.inc.

```
00007   {
00008     for(int i=0; i<NumV; ++i)
00009       {
00010         D[i] = INF;
00011         if(P) P[i] = -1;
00012       }
00013
```

4.62.2.22 template<class DT, class WT> int sdGraph< DT, WT >::Input (istream & *is*) [inline]

Definition at line 880 of file sdgraph.inc.

```
00881   { return 1; }
00882
```

4.62.2.23 template<class DT, class WT> int sdGraph< DT, WT >::InputBinary (istream & *is*) [inline]

Definition at line 769 of file sdgraph.inc.

Referenced by operator>=().

```
00770   {
00771     // Delete everything.
00772     //
00773     Clear();
00774 
00775     // Read in "NumV":
00776     //
00777     is >= NumV;
00778 
00779     // Read in "NumE":
00780     //
00781     is >= NumE;
00782 
00783     int vid;
00784     // Read in "VA":
00785     //
00786     if(NumV>0)
00787       {
00788         // Create the vertices:
00789         //
00790         if(!(VA = new sdVertex<WT>*[NumV]))
00791             { cerr << "sdGraph::InputBinary#1\n"; raise(SIGQUIT); }
00792         for(int i=0; i<NumV; ++i)
00793           {
00794             if(!(VA[i] = new sdVertex<WT>))
00795               { cerr << "sdGraph::InputBinary#2\n"; raise(SIGQUIT); }
00796             VA[i]->ID = i;
00797             VA[i]->NumO = 0;
00798             VA[i]->NumI = 0;
00799             VA[i]->OutE = 0;
00800             VA[i]->InE = 0;
00801           }
00802 
00803         for(int i=0; i<NumV; ++i)
00804           {
00805             // Read in number of out-edges:
00806             //
00807             is >= VA[i]->NumO;
00808 
00809             // Read in number of in-edges:
00810             //
00811             is >= VA[i]->NumI;
00812 
00813             if(VA[i]->NumO>0 &&
00814                !(VA[i]->OutE = new sdEdge<WT>[VA[i]->NumO]))
00815                 { cerr << "sdGraph::InputBinary#3\n"; raise(SIGQUIT); }
00816 
00817             if(VA[i]->NumI>0 &&
00818                !(VA[i]->InE = new sdEdge<WT>[VA[i]->NumI]))
00819                 { cerr << "sdGraph::InputBinary#4\n"; raise(SIGQUIT); }
00820 
00821             // Read in out-edges:
00822             //
00823             if(VA[i]->NumO>0)
00824               {
00825                 for(int j=0; j<VA[i]->NumO; ++j)
00826                   {
00827                     // Read in the id and point to the correct vertex:
```

```
00828                     //
00829                     is >= vid;
00830
00831                     if(vid < 0 || vid>=NumV)
00832                       { cerr << "sdGraph::InputBinary#5\n"; raise(SIGQUIT); }
00833
00834                     VA[i]->OutE[j].V = VA[vid];
00835
00836                     // Read in the weight of the edge:
00837                     //
00838                     is >= VA[i]->OutE[j].W;
00839                   }
00840               }
00841
00842             // Read in in-edges:
00843             //
00844             if(VA[i]->NumI>0)
00845               {
00846                 for(int j=0; j<VA[i]->NumI; ++j)
00847                   {
00848                     // Read in the id and point to the correct vertex:
00849                     //
00850                     is >= vid;
00851
00852                     if(vid < 0 || vid>=NumV)
00853                       { cerr << "sdGraph::InputBinary#6\n"; raise(SIGQUIT); }
00854
00855                     VA[i]->InE[j].V = VA[vid];
00856
00857                     // Read in the weight of the edge:
00858                     //
00859                     is >= VA[i]->InE[j].W;
00860                   }
00861               }
00862
00863           }
00864       }
00865
00866     // Read in "Data":
00867     //
00868     if(NumV>0)
00869       {
00870         if(!(Data = new DT[NumV]))
00871             { cerr << "sdGraph::InputBinary#7\n"; raise(SIGQUIT); }
00872         for(int i=0; i<NumV; ++i)
00873             is >= Data[i];
00874       }
00875
```

4.62.2.24 template<class DT, class WT> int sdGraph< DT, WT >::NumEdges () const [inline]

Definition at line 346 of file sdgraph.inc.

```
00347   { return NumE; }
00348
```

4.62.2.25 template<class DT, class WT> int sdGraph< DT, WT >::NumSCC () const [inline]

Definition at line 357 of file sdgalg.inc.

```
00358   { return SCC().Number(); }
00359
```

4.62.2.26 template<class DT, class WT> int sdGraph< DT, WT >::NumVertices () const [inline]

Definition at line 342 of file sdgraph.inc.

Referenced by ConstraintManager::Number().

```
00343   { return NumV; }
00344
```

4.62.2.27 template<class DT, class WT = double> int sdGraph<DT, WT>::OutDegree (const sdVertex< WT > * *vert*) const [private]

4.62.2.28 template<class DT, class WT> int sdGraph< DT, WT >::OutDegree (int *i*) const [inline]

Definition at line 376 of file sdgraph.inc.

```
00377   {
00378     if(!ValidVertex(i))
00379       { cerr << "sdGraph::OutDegree\n"; raise(SIGQUIT); }
00380
```

4.62.2.29 template<class DT, class WT> int sdGraph< DT, WT >::Output (ostream & *os*) const [inline]

Definition at line 604 of file sdgraph.inc.

```
00605   {
00606     // Print the data, ID, and the number of outgoing/incoming edges
00607     // for each vertex.
00608     //
00609     os << V_BEGIN_DELIM << '\n';
00610     for(int v=0; v<NumV; ++v)
00611         os << '\t' << v << '\t' << Data[v] << '\n';
00612     os << V_END_DELIM << '\n';
00613
00614     // Print the edges of the vertices
00615     //
00616     os << E_BEGIN_DELIM << '\n';
00617     for(int v=0; v<NumV; ++v)
00618       {
00619         if(VA[v]) VA[v]->PrintOutEdges(os);
00620       }
00621
```

4.62.2.30 template<class DT, class WT> int sdGraph< DT, WT >::OutputBinary (ostream & *os*) const [inline]

Definition at line 884 of file sdgraph.inc.

```
00885   {
00886     // Write "NumV":
00887     //
00888     os <= NumV;
00889
00890     // Write "NumE":
00891     //
00892     os <= NumE;
00893
00894     // Write "VA":
00895     //
00896     if(VA && NumV>0)
00897       {
00898         for(int i=0; i<NumV; ++i)
00899           {
00900             // Write the number of out edges.
00901             //
00902             os <= VA[i]->NumO;
00903
00904             // Write the number of in edges.
00905             //
00906             os <= VA[i]->NumI;
00907
00908             // Write the out edges (id then weight).
00909             //
00910             if(VA[i]->OutE && VA[i]->NumO > 0)
00911               {
00912                 for(int j=0; j<VA[i]->NumO; ++j)
00913                   {
00914                     os <= VA[i]->OutE[j].V->ID;
00915                     os <= VA[i]->OutE[j].W;
00916                   }
00917               }
00918
00919             // Write the in edges (id then weight).
00920             //
00921             if(VA[i]->InE && VA[i]->NumI > 0)
00922               {
00923                 for(int j=0; j<VA[i]->NumI; ++j)
00924                   {
00925                     os <= VA[i]->InE[j].V->ID;
00926                     os <= VA[i]->InE[j].W;
00927                   }
00928               }
00929           }
00930       }
00931
00932     // Write "Data":
00933     //
00934     if(Data && NumV>0)
00935       {
00936         for(int i=0; i<NumV; ++i)
00937             os <= Data[i];
00938       }
00939
```

4.62.2.31 template<class DT, class WT> ostream & sdGraph< DT, WT >::PrintDotty (ostream & *os*, int *edges* = 0) const [inline]

Definition at line 626 of file sdgraph.inc.

Referenced by ConstraintManager::PrintGraph().

```
00627   {
00628     int to;
00629     WT w;
00630
00631     os << "digraph G {\n";
00632     os << "\tcenter = true;\n";
00633     os << "\tratio = compress;\n";
00634     os << "\tsize =\"7.5,10\";\n";
00635     // Define all the labels for the vertices:
00636     for(int i=0; i<NumV; ++i)
00637       {
00638         // GRAPE customization
00639         os << '\t' << i+1 << " [label = \"" << Data[i]->Label()  << "\"];\n";
00640         //      os << '\t' << i+1 << " [label = \"" << Data[i]   << "\"];\n";
00641       }
00642
00643     // Print all the edges:
00644     for(int i=0; i<NumV; ++i)
00645       {
00646         for(int j=0; j<VA[i]->NumO; ++j)
00647           {
00648             to = VA[i]->OutE[j].V->ID;
00649             w  = VA[i]->OutE[j].W;
00650             os << '\t' << i+1 << " -> " << to+1;
00651             if(edges)
00652                 os << " [label=\"" << w << "\"]";
00653             os << ";\n";
00654           }
00655       }
00656     os << "}
00657
```

4.62.2.32 template<class DT, class WT> void sdGraph< DT, WT >::Relax (int *u*, int *v*, jVector & *D*, int * *P*, int *rev*, int *wc*) const [inline, private]

Definition at line 18 of file sdgalg.inc.

```
00022   {
00023     WT w = (wc==0) ? WT(1.0) : (
00024                (wc>0)  ? (!rev ? GetWeight(u,v,0) : GetWeight(v,u,0)) :
00025                          (!rev ? GetWeight(u,v,1) : GetWeight(v,u,1))
00026                                );
00027     if(D[v] > D[u] + ((double)w))
00028       {
00029         D[v] = D[u] + ((double)w);
00030         if(P) P[v] = u;
00031       }
00032
```

4.62.2.33 template<class DT, class WT> SDG_SetSetInt sdGraph< DT, WT >::SCC () const [inline]

Definition at line 126 of file sdgalg.inc.

Referenced by NumSCC(), SCCGraph(), and StronglyConnectedComponents().

```
00127   {
00128     SDG_SetSetInt scc;
00129     SDG_SetInt component;
00130     int u, * finish=0, * invfin = 0;
```

```
00131
00132     if(NumV<1)
00133         return scc;
00134
00135     if(!(finish = new int[NumV]))
00136         raise(SIGQUIT);
00137     if(!(invfin = new int[NumV]))
00138         raise(SIGQUIT);
00139
00140     DFTimeStamps(0,finish,0);
00141
00142     for(u=0; u<NumV; ++u)
00143         invfin[ finish[u] ] = u;
00144
00145     // Color all the vertices white to mark them as unvisited:
00146     //
00147     sdg_color color[NumV];
00148     for(u=0; u<NumV; ++u)
00149         color[u] = sdg_white;
00150
00151     for(int i=NumV-1; i>=0; --i)
00152       {
00153         u = invfin[i];
00154         if(color[u] == sdg_white)
00155           {
00156             DF_Visit(u, color, 0, 0, 0, &component, 1);
00157             // scc.Add(component.UniqueCopy());
00158             scc.insert(component);
00159             component.clear();
00160           }
00161       }
00162
```

4.62.2.34 template<class DT, class WT> sdGraph< DT, WT > sdGraph< DT, WT >::SCCGraph () const [inline]

Definition at line 172 of file sdgalg.inc.

```
00173   {
00174     sdGraph<DT,WT> sccgraph;
00175     SDG_SetSetInt sccset = SCC();
00176     SDG_SetSetInt::const_iterator it1;
00177     SDG_SetInt::const_iterator    it2;
00178     int * v2c = 0;              // Mapping from vertex id to component number.
00179     int * numoe = 0;            // Number of outbound edges for each component.
00180     int * numie = 0;            // Number of inbound edges for each component.
00181     int ** OutEdge=0;           // Adjacency matrix for outbound edges.
00182     int ** InEdge=0;            // Adjacency matrix for inbound edges.
00183     int u,v,e;                  // Vertex id's and edge iterator.
00184     int c=0;                    // Increments through components.
00185
00186     // cerr << "NumV = " << NumV << '\n';
00187     // cerr << "NumC = " << sccset.Number() << '\n';
00188     // cerr << "NumC = " << sccset.size() << '\n';
00189     // raise(SIGQUIT);
00190
00191     if(NumV < 1)
00192         return sccgraph;
00193
00194     // if(sccset.Number() == 1)
00195     if(sccset.size() == 1)
00196       {
```

```
00197         sccgraph.NumV = 1;
00198         if(!(sccgraph.VA = new sdVertex<WT>*[1]) ||
00199            !(sccgraph.VA[0] = new sdVertex<WT>))
00200             raise(SIGQUIT);
00201         if(!(sccgraph.Data = new DT[1]))
00202             raise(SIGQUIT);
00203         sccgraph.VA[0]->ID = 0;
00204         sccgraph.Data[0] = Data[0];
00205         return sccgraph;
00206       }
00207 
00208     // sccgraph.NumV = sccset.Number(); // Number of strongly conn. components.
00209     sccgraph.NumV = sccset.size();      // Number of strongly conn. components.
00210 
00211     // Allocate space.
00212     if(!(v2c = new int[NumV]))
00213         raise(SIGQUIT);
00214     if(!(numoe = new int[sccgraph.NumV]))
00215         raise (SIGQUIT);
00216     if(!(numie = new int[sccgraph.NumV]))
00217         raise (SIGQUIT);
00218     if(!(sccgraph.VA = new sdVertex<WT>*[sccgraph.NumV]))
00219         raise(SIGQUIT);
00220     if(!(sccgraph.Data = new DT[sccgraph.NumV]))
00221         raise(SIGQUIT);
00222     if(!(OutEdge = new int*[sccgraph.NumV]))
00223         raise(SIGQUIT);
00224     if(!(InEdge = new int*[sccgraph.NumV]))
00225         raise(SIGQUIT);
00226 
00227     // Initialize the adjacency matrices.
00228     for(u=0; u<sccgraph.NumV; ++u)
00229       {
00230         if(!(OutEdge[u] = new int[sccgraph.NumV]))
00231             raise(SIGQUIT);
00232         if(!(InEdge[u] = new int[sccgraph.NumV]))
00233             raise(SIGQUIT);
00234 
00235         for(v=0; v<sccgraph.NumV; ++v)
00236           {
00237             OutEdge[u][v] = 0;
00238             InEdge[u][v]  = 0;
00239           }
00240         numoe[u] = 0;
00241         numie[u] = 0;
00242       }
00243 
00244     // Initialize the mapping from vertices to components.
00245     for(u=0; u<NumV; ++u)
00246         v2c[u] = -1;
00247 
00248     // Introduce vertices into the SCC-Graph.
00249     c=0;
00250     int got;
00251     for(it1 = sccset.begin(); it1!=sccset.end(); ++it1)
00252       {
00253         if(!(sccgraph.VA[c] = new sdVertex<WT>))
00254             raise(SIGQUIT);
00255         sccgraph.VA[c]->ID = c;
00256 
00257         // Record the vertices that belong to this component.
00258         got = 0;
00259         for(it2 = ($\ast$it1).begin(); it2!=($\ast$it1).end(); ++it2)
00260           {
00261             v2c[*it2] = c;
```

```
00262
00263            // Copy one of the vertices' data into the component graph.
00264            if(!got)
00265              {
00266                sccgraph.Data[c] = Data[*it2];
00267                got = 1;
00268              }
00269          }
00270        ++c;
00271      }
00272
00273    // Make sure the mapping from vertices to components is complete:
00274    //
00275    for(u=0; u<NumV; ++u)
00276        if(v2c[u] == -1)
00277            raise(SIGQUIT);
00278
00279    // Create the adjacency matrices.
00280    for(u=0; u<NumV; ++u)
00281      {
00282        // Find out which component this vertex belongs to.
00283        c = v2c[u];
00284
00285        // Record the outbound component edges in the adjacency matrix.
00286        for(e=0; e<VA[u]->NumO; ++e)
00287          {
00288            v = VA[u]->OutE[e].V->ID;
00289            if((v2c[v] != c) && !OutEdge[c][v2c[v]])
00290              {
00291                OutEdge[c][v2c[v]] = 1;
00292                ++numoe[c];
00293              }
00294          }
00295
00296        // Record the inbound component edges in the adjacency matrix.
00297        for(e=0; e<VA[u]->NumI; ++e)
00298          {
00299            v = VA[u]->InE[e].V->ID;
00300            if((v2c[v] != c) && !InEdge[c][v2c[v]])
00301              {
00302                InEdge[c][v2c[v]] = 1;
00303                ++numie[c];
00304              }
00305          }
00306      }
00307
00308    // Make the edges in the component graph.
00309    for(u=0; u<sccgraph.NumV; ++u)
00310      {
00311        sccgraph.VA[u]->NumO = numoe[u];
00312        sccgraph.VA[u]->NumI = numie[u];
00313
00314        // Create space for the edges.
00315        if((numoe[u]>0) && !(sccgraph.VA[u]->OutE = new sdEdge<WT>[numoe[u]]))
00316            raise(SIGQUIT);
00317        if((numie[u]>0) && !(sccgraph.VA[u]->InE = new sdEdge<WT>[numie[u]]))
00318            raise(SIGQUIT);
00319
00320        // Read which edges to introduce from the adjacency matrix.
00321        c = 0;
00322        for(v=0; v<sccgraph.NumV; ++v)
00323          {
00324            if(OutEdge[u][v])
00325              {
00326                sccgraph.VA[u]->OutE[c].V = sccgraph.VA[v];
```

```
00327                 sccgraph.VA[u]->OutE[c].W = 1.0;
00328                 ++c;
00329               }
00330           }
00331         c = 0;
00332         for(v=0; v<sccgraph.NumV; ++v)
00333           {
00334             if(InEdge[u][v])
00335               {
00336                 sccgraph.VA[u]->InE[c].V = sccgraph.VA[v];
00337                 sccgraph.VA[u]->InE[c].W = 1.0;
00338                 ++c;
00339               }
00340           }
00341       }
00342
00343     delete[] v2c;
00344     delete[] numoe;
00345     delete[] numie;
00346     for(u=0; u<sccgraph.NumV; ++u)
00347       {
00348         delete[] OutEdge[u];
00349         delete[] InEdge[u];
00350       }
00351
```

4.62.2.35 template<class DT, class WT> SDG_SetSetInt sdGraph< DT, WT >::StronglyConnectedComponents () const [inline]

Definition at line 168 of file sdgalg.inc.

```
00169   { return SCC(); }
00170
```

4.62.2.36 template<class DT, class WT> sdGraph< DT, WT > sdGraph< DT, WT >::SubGraph (const SDG_SetInt & *vertset*) const [inline]

Definition at line 476 of file sdgraph.inc.

```
00477   {
00478     sdGraph<DT,WT> sub;
00479     int * subid=0;              // Maps id's from this graph to the subgraph.
00480     int * vertex=0;             // Array of verts (copied from vertset).
00481     int u,v,e,i,nbr,numv;
00482     // SDG_SetInt::ConstIterator it;
00483     SDG_SetInt::const_iterator it;
00484
00485     if(NumV<1) return sub;
00486
00487     // Filter out any bad vertex indices from the set:
00488     numv=0;
00489     // for(it = vertset.GetConstIterator(); it; ++it)
00490     for(it = vertset.begin(); it!=vertset.end(); ++it)
00491         if(($\ast$it)>=0 && ($\ast$it)<NumV)
00492             ++numv;
00493
00494     if(numv<1) return sub;
00495
00496     // Make space for the vertices and the data.
00497     if(!(sub.VA = new sdVertex<WT>*[numv]))
```

```
00498       { cerr << "sdGraph::SubGraph#1\n"; raise(SIGQUIT); }
00499     if(!(sub.Data = new DT[numv]))
00500       { cerr << "sdGraph::SubGraph#2\n"; raise(SIGQUIT); }
00501
00502     sub.NumV = numv;
00503
00504     // Copy the vertices into a static array.
00505     if(!(vertex = new int[sub.NumV]))
00506       { cerr << "sdGraph::SubGraph#3\n"; raise(SIGQUIT); }
00507     u=0;
00508     // for(it.Reset(); it; ++it)
00509     for(it = vertset.begin(); it!=vertset.end(); ++it)
00510       {
00511         v = *it;
00512         if(v>=0 && v<NumV)
00513             vertex[u++] = v;
00514       }
00515
00516     // Sort these vertices to make sure in increasing order.
00517     QuickSort(vertex,sub.NumV);
00518
00519     if(!(subid = new int[NumV]))
00520       { cerr << "sdGraph::SubGraph#4\n"; raise(SIGQUIT); }
00521
00522     for(v=0; v<NumV; ++v)       // Mark the id's as not existing in subgraph
00523         subid[v] = -1;
00524
00525     // Determine the vertices from this graph to be included in the subgraph.
00526     for(u=0; u<sub.NumV; ++u)
00527       {
00528         v = vertex[u];
00529         if(!(sub.VA[u] = new sdVertex<WT>))
00530           { cerr << "sdGraph::SubGraph#5\n"; raise(SIGQUIT); }
00531         sub.VA[u]->ID = u;
00532         subid[v] = u;
00533
00534         // Copy the data in.
00535         sub.Data[u] = Data[v];
00536       }
00537
00538     // Introduce the edges into the subgraph.
00539     for(u=0; u<sub.NumV; ++u)
00540       {
00541         v = vertex[u];
00542
00543         // Determine the number of outbound edges for vertex u.
00544         sub.VA[u]->NumO = 0;
00545         for(e=0; e<VA[v]->NumO; ++e)
00546           {
00547             nbr = subid[ VA[v]->OutE[e].V->ID ];
00548             if(nbr != -1)
00549                 ++(sub.VA[u]->NumO);
00550           }
00551
00552         // Determine the number of inbound edges for vertex u.
00553         sub.VA[u]->NumI = 0;
00554         for(e=0; e<VA[v]->NumI; ++e)
00555           {
00556             nbr = subid[ VA[v]->InE[e].V->ID ];
00557             if(nbr != -1)
00558                 ++(sub.VA[u]->NumI);
00559           }
00560
00561         // Allocate space for the edges.
00562         if(sub.VA[u]->NumO>0 &&
```

```
00563           !(sub.VA[u]->OutE = new sdEdge<WT>[sub.VA[u]->NumO]))
00564             { cerr << "sdGraph::SubGraph#6\n"; raise(SIGQUIT); }
00565         if(sub.VA[u]->NumI>0 &&
00566           !(sub.VA[u]->InE = new sdEdge<WT>[sub.VA[u]->NumI]))
00567             { cerr << "sdGraph::SubGraph#7\n"; raise(SIGQUIT); }
00568
00569         // Copy the outbound edges.
00570         i=0;
00571         for(e=0; e<VA[v]->NumO; ++e)
00572           {
00573             nbr = subid[ VA[v]->OutE[e].V->ID ];
00574             if(nbr != -1)
00575               {
00576                 sub.VA[u]->OutE[i].V = sub.VA[nbr];
00577                 sub.VA[u]->OutE[i].W = VA[v]->OutE[e].W;
00578                 ++i;
00579                 ++(sub.NumE);
00580               }
00581           }
00582
00583         // Copy the inbound edges.
00584         i=0;
00585         for(e=0; e<VA[v]->NumI; ++e)
00586           {
00587             nbr = subid[ VA[v]->InE[e].V->ID ];
00588             if(nbr != -1)
00589               {
00590                 sub.VA[u]->InE[i].V = sub.VA[nbr];
00591                 sub.VA[u]->InE[i].W = VA[v]->InE[e].W;
00592                 ++i;
00593               }
00594           }
00595         cerr << "]\n";
00596       }
00597
```

4.62.2.37 template<class DT, class WT> SDG_IntList ** sdGraph< DT, WT >::TravLists (int *src*, int *k*, KAdjMat * *P* = 0) const [inline]

Definition at line 706 of file sdgraph.inc.

Referenced by begin_path().

```
00707   {
00708     if(NumV<1 || k<1 || src<0 || src>=NumV) return 0;
00709
00710     DLList<int> ** T;   // The traversal lists, originating at the source.
00711     int alloc = 0;      // Keeps track if we allocate space for P.
00712
00713     if(!(T = new DLList<int>*[k-1])) return 0;
00714     for(int i=0; i<k; ++i)
00715       {
00716         if(!(T[i] = new DLList<int>[NumV]))
00717           {
00718             for(int j=0; j<i; ++j)
00719                 delete[] T[j];
00720             delete[] T;
00721             return 0;
00722           }
00723       }
00724
00725     if(!P)
00726       // Have to compute the the path matrices (0...k-1):
```

```
00727       {
00728         if(!(P = new KAdjMat[k]))
00729           { delete[] T; return 0; }
00730         P[0] = AdjMatrix();
00731 
00732         for(int i=1; i<k; ++i) P[i] = P[i-1]*P[0];
00733         alloc = 1;
00734       }
00735 
00736     ABS a;
00737     for(int i=0; i<k-1; ++i)
00738       {
00739         for(int j=0; j<NumV; ++j)
00740           {
00741             a = P[i][src]&P[0](j);
00742             for(int l=0; l<NumV; ++l)
00743               {
00744                 if(a.IsSet((uint) l))
00745                     T[i][j].AddToBack(l);
00746               }
00747           }
00748       }
00749 
```

4.62.2.38 template<class DT, class WT> KAdjMat * sdGraph< DT, WT >::TravMatrices (int *src*, int *k*, KAdjMat * *P* = 0) const [inline]

Definition at line 675 of file sdgraph.inc.

```
00676   {
00677     if(NumV<1 || k<1 || src<0 || src>=NumV) return 0;
00678 
00679     KAdjMat * T;        // The traversal matrices, originating at the source.
00680     int alloc = 0;      // Keeps track if we allocate space for P.
00681 
00682     KAdjMat::DefaultSize(NumV);
00683     if(!(T = new KAdjMat[k-1])) return 0;
00684     if(!P)
00685       // Have to compute the the path matrices (0...k-1):
00686       {
00687         if(!(P = new KAdjMat[k]))
00688           { delete[] T; return 0; }
00689         P[0] = AdjMatrix();
00690 
00691         for(int i=1; i<k; ++i) P[i] = P[i-1]*P[0];
00692         alloc = 1;
00693       }
00694 
```

4.62.2.39 template<class DT, class WT> int sdGraph< DT, WT >::ValidData (int *i*) const [inline, private]

Definition at line 372 of file sdgraph.inc.

Referenced by GetData().

```
00373   { return (Data && i>=0 && i<NumV); }
00374 
```

4.62.2.40 template<class DT, class WT> int sdGraph< DT, WT >::ValidVertex (int *i*) const [inline, private]

Definition at line 368 of file sdgraph.inc.

Referenced by Degree(), GetVertex(), GetWeight(), InDegree(), and OutDegree().

```
00369   { return (VA && i>=0 && i<NumV && VA[i]); }
00370
```

4.62.2.41 template<class DT, class WT = double> bf_iterator sdGraph<DT, WT>::begin_bf (int *vndx* = 0, int *rev* = 0) [inline]

Definition at line 210 of file sdgraph.h.

```
00211     { return bf_iterator(this, vndx, rev); }
00212
```

4.62.2.42 template<class DT, class WT = double> df_iterator sdGraph<DT, WT>::begin_df (int *vndx* = 0, int *rev* = 0) [inline]

Definition at line 208 of file sdgraph.h.

```
00209     { return df_iterator(this, vndx, rev); }
00210
```

4.62.2.43 template<class DT, class WT = double> neighbor_iterator sdGraph<DT, WT>::begin_neighbor (int *vndx* = 0, int *rev* = 0) [inline]

Definition at line 206 of file sdgraph.h.

Referenced by ConstraintManager::getParents().

```
00207     { return neighbor_iterator(this, vndx, rev); }
00208
```

4.62.2.44 template<class DT, class WT = double> path_iterator sdGraph<DT, WT>::begin_path (int *src*, int *tgt*, int *len*, DLList< int > *const * *tl* = 0, int(* *filter*)(const sdGraph< DT, WT > &graph, const int *path, int ndx, int pathlen, void *arbitrary_criteria) = 0, void * *criteria* = 0) [inline]

Definition at line 212 of file sdgraph.h.

```
00219     {
00220       if(tl)
00221       return path_iterator((const sdGraph<DT,WT>*) this,
00222                            src, tgt,
00223                            len, tl, filter, criteria);
00224       path_iterator pi;
00225
00226       //
00227       DLList<int> * const * trav = TravLists(src,len);
00228       pi   = path_iterator((const sdGraph<DT,WT>*) this, src, tgt,
```

```
00229                           len, trav, filter, criteria);
00230
00231       for(int i=0; i<len-1; ++i) delete[] trav[i];
00232       delete[] trav;
00233       //
00234       return pi;
00235
```

4.62.2.45 template<class DT, class WT> WT & sdGraph< DT, WT >::operator() (int *i*, int *j*, int *alt* = 0) [inline]

Definition at line 472 of file sdgraph.inc.

```
00473   { return GetWeight(i,j,alt); }
00474
```

4.62.2.46 template<class DT, class WT> const WT & sdGraph< DT, WT >::operator() (int *i*, int *j*, int *alt* = 0) const [inline]

Definition at line 468 of file sdgraph.inc.

```
00469   { return GetWeight(i,j,alt); }
00470
```

4.62.2.47 template<class DT, class WT> DT & sdGraph< DT, WT >::operator() (int *i*) [inline]

Definition at line 444 of file sdgraph.inc.

```
00445   { return GetData(i); }
00446
```

4.62.2.48 template<class DT, class WT> const DT & sdGraph< DT, WT >::operator() (int *i*) const [inline]

Definition at line 440 of file sdgraph.inc.

```
00441   { return GetData(i); }
00442
```

4.62.2.49 template<class DT, class WT> const sdGraph< DT, WT > & sdGraph< DT, WT >::operator= (const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 304 of file sdgraph.inc.

```
00305   { Copy(ddg, true); return *this; }
00306
```

4.62.2.50 template<class DT, class WT> const sdGraph< DT, WT > & sdGraph< DT, WT >::operator= (const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 300 of file sdgraph.inc.

```
00301   { Copy(sdg); return *this; }
00302
```

4.62.2.51 template<class DT, class WT> sdVertex< WT > & sdGraph< DT, WT >::operator[] (int *i*) [inline]

Definition at line 420 of file sdgraph.inc.

```
00421   { return GetVertex(i); }
00422
```

4.62.2.52 template<class DT, class WT> const sdVertex< WT > & sdGraph< DT, WT >::operator[] (int *i*) const [inline]

Definition at line 416 of file sdgraph.inc.

```
00417   { return GetVertex(i); }
00418
```

4.62.3 Friends And Related Function Documentation

4.62.3.1 template<class DT, class WT = double> class ddGraph [friend]

Definition at line 75 of file sdgraph.h.

4.62.3.2 template<class DT, class WT = double> ostream& operator (ostream & *os*, const sdGraph< DT, WT > & *sdg*) [friend]

4.62.3.3 template<class DT, class WT = double> istream& operator>= (istream & *is*, sdGraph< DT, WT > & *sdg*) [friend]

Definition at line 944 of file sdgraph.inc.

```
00945   { sdg.InputBinary(is); return is; }
00946
```

4.62.4 Member Data Documentation

4.62.4.1 template<class DT, class WT = double> DT * sdGraph<DT, WT>::Data [private]

Definition at line 296 of file sdgraph.h.

4.62.4.2 template<class DT, class WT = double> int sdGraph<DT, WT>::NumE [private]

Definition at line 290 of file sdgraph.h.

4.62.4.3 template<class DT, class WT = double> int sdGraph<DT, WT>::NumV [private]

Definition at line 289 of file sdgraph.h.

4.62.4.4 template<class DT, class WT = double> sdVertex< WT > ** sdGraph<DT, WT>::VA [private]

Definition at line 293 of file sdgraph.h.

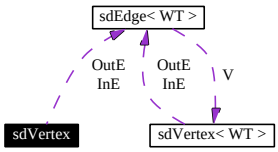
The documentation for this class was generated from the following files:

- **sdgraph.h**
- **sdgalg.inc**
- **sdgraph.inc**

4.63 sdVertex Class Template Reference

`#include <sdvertex.h>`

Collaboration diagram for sdVertex:



Public Methods

- **sdVertex** ()
- **sdVertex** (const sdVertex< WT > &sdv)
- **sdVertex** (int id, int num_out, int num_in)
- **~sdVertex** ()
- int **OutDegree** () const
- int **InDegree** () const
- int **Degree** () const
- void **SortEdges** ()
- void **PrintOutEdges** (ostream &os) const
- void **PrintInEdges** (ostream &os) const
- void **PrintEdges** (ostream &os) const
- const sdVertex<WT>& **operator=** (const sdVertex< WT > &sdv)
- int **Compare** (const sdVertex< WT > &v) const
- **MAKECOMPARISONS_WITH_COMPARE** (sdVertex< WT >)
- const sdVertex<WT>& **GetOutNeighbor** (int i) const
- sdVertex<WT>& **GetOutNeighbor** (int i)
- const sdVertex<WT>& **operator[]** (int i) const
- sdVertex<WT>& **operator[]** (int i)
- const sdVertex<WT>& **GetInNeighbor** (int i) const
- sdVertex<WT>& **GetInNeighbor** (int i)
- const sdVertex<WT>& **operator()** (int i) const
- sdVertex<WT>& **operator()** (int i)
- int **GetOutEdgeIndex** (const sdVertex< WT > &to) const
- int **GetOutEdgeIndex** (int to_id) const
- int **GetInEdgeIndex** (const sdVertex< WT > &from) const
- int **GetInEdgeIndex** (int from_id) const
- int **HasEdgeTo** (const sdVertex< WT > &to) const
- int **HasEdgeFrom** (const sdVertex< WT > &from) const
- const **sdEdge**<WT>& **GetOutEdge** (int i) const
- **sdEdge**<WT>& **GetOutEdge** (int i)
- const **sdEdge**<WT>& **GetInEdge** (int i) const
- **sdEdge**<WT>& **GetInEdge** (int i)
- const WT& **GetOutWeight** (int i) const
- WT& **GetOutWeight** (int i)
- const WT& **GetInWeight** (int i) const
- WT& **GetInWeight** (int i)

Public Attributes

- int **ID**
- int **NumO**
- int **NumI**
- **sdEdge**<WT>* **OutE**
- **sdEdge**<WT>* **InE**

Friends

- ostream& **operator** (ostream &os, const sdVertex< WT > &vert)
- int **Connected** (const sdVertex< WT > &v1, const sdVertex< WT > &v2)

template<class WT = double> class sdVertex

4.63.1 Constructor & Destructor Documentation

4.63.1.1 template<class WT = double> sdVertex<WT>::sdVertex<WT> ()

4.63.1.2 template<class WT = double> sdVertex<WT>::sdVertex<WT> (const sdVertex< WT > & *sdv*)

4.63.1.3 template<class WT = double> sdVertex<WT>::sdVertex<WT> (int *id*, int *num_out*, int *num_in*)

4.63.1.4 template<class WT> sdVertex< WT >::~sdVertex<WT> () [inline]

Definition at line 81 of file sdvertex.inc.

```
00082   { delete[] OutE; delete[] InE; }
00083
```

4.63.2 Member Function Documentation

4.63.2.1 template<class WT> int sdVertex< WT >::Compare (const sdVertex< WT > & *v*) const [inline]

Definition at line 170 of file sdvertex.inc.

```
00171   { return (ID - v.ID); }
00172
```

4.63.2.2 template<class WT> int sdVertex< WT >::Degree () const [inline]

Definition at line 93 of file sdvertex.inc.

```
00094   { return OutDegree() + InDegree(); }
00095
```

4.63.2.3 template<class WT> sdEdge< WT > & sdVertex< WT >::GetInEdge (int *i*) [inline]

Definition at line 260 of file sdvertex.inc.

```
00261   { return InE[i]; }
00262
```

4.63.2.4 template<class WT> const sdEdge< WT > & sdVertex< WT >::GetInEdge (int *i*) const [inline]

Definition at line 256 of file sdvertex.inc.

```
00257   { return InE[i]; }
00258
```

4.63.2.5 template<class WT> int sdVertex< WT >::GetInEdgeIndex (int *from_id*) const [inline]

Definition at line 221 of file sdvertex.inc.

```
00222   {
00223     if(!NumI) return -1;
00224
00225     sdEdge<WT> edge; edge.V = new sdVertex<WT>(from_id, 0, 0);
00226     int index = BinSearch(InE, NumI, edge);
00227     delete edge.V;
00228     return index;
00229
```

4.63.2.6 template<class WT> int sdVertex< WT >::GetInEdgeIndex (const sdVertex< WT > & *from*) const [inline]

Definition at line 232 of file sdvertex.inc.

Referenced by HasEdgeFrom().

```
00233   { return GetInEdgeIndex(from.ID); }
00234
```

4.63.2.7 template<class WT> sdVertex< WT > & sdVertex< WT >::GetInNeighbor (int *i*) [inline]

Definition at line 194 of file sdvertex.inc.

```
00195   { return *(InE[i].V); }
00196
```

4.63.2.8 template<class WT> const sdVertex< WT > & sdVertex< WT >::GetInNeighbor (int *i*) const [inline]

Definition at line 190 of file sdvertex.inc.

Referenced by operator()().

```
00191   { return *(InE[i].V); }
00192
```

4.63.2.9 template<class WT> WT & sdVertex< WT >::GetInWeight (int *i*) [inline]

Definition at line 276 of file sdvertex.inc.

```
00277   { return InE[i].W; }
00278
```

4.63.2.10 template<class WT> const WT & sdVertex< WT >::GetInWeight (int *i*) const [inline]

Definition at line 272 of file sdvertex.inc.

Referenced by sdGraph::GetWeight().

```
00273   { return InE[i].W; }
00274
```

4.63.2.11 template<class WT> sdEdge< WT > & sdVertex< WT >::GetOutEdge (int *i*) [inline]

Definition at line 252 of file sdvertex.inc.

```
00253   { return OutE[i]; }
00254
```

4.63.2.12 template<class WT> const sdEdge< WT > & sdVertex< WT >::GetOutEdge (int *i*) const [inline]

Definition at line 248 of file sdvertex.inc.

```
00249   { return OutE[i]; }
00250
```

4.63.2.13 template<class WT> int sdVertex< WT >::GetOutEdgeIndex (int *to_id*) const [inline]

Definition at line 206 of file sdvertex.inc.

```
00207   {
00208     if(!NumO) return -1;
00209
00210     sdEdge<WT> edge; edge.V = new sdVertex<WT>(to_id, 0, 0);
00211     int index = BinSearch(OutE, NumO, edge);
00212     delete edge.V;
00213     return index;
00214
```

4.63.2.14 template<class WT> int sdVertex< WT >::GetOutEdgeIndex (const sdVertex< WT > & *to*) const [inline]

Definition at line 217 of file sdvertex.inc.

Referenced by HasEdgeTo().

```
00218   { return GetOutEdgeIndex(to.ID); }
00219
```

4.63.2.15 template<class WT> sdVertex< WT > & sdVertex< WT >::GetOutNeighbor (int *i*) [inline]

Definition at line 178 of file sdvertex.inc.

```
00179   { return *(OutE[i].V); }
00180
```

4.63.2.16 template<class WT> const sdVertex< WT > & sdVertex< WT >::GetOutNeighbor (int *i*) const [inline]

Definition at line 174 of file sdvertex.inc.

Referenced by operator[]().

```
00175   { return *(OutE[i].V); }
00176
```

4.63.2.17 template<class WT> WT & sdVertex< WT >::GetOutWeight (int *i*) [inline]

Definition at line 268 of file sdvertex.inc.

```
00269   { return OutE[i].W; }
00270
```

4.63.2.18 template<class WT> const WT & sdVertex< WT >::GetOutWeight (int *i*) const [inline]

Definition at line 264 of file sdvertex.inc.

Referenced by sdGraph::GetWeight().

```
00265   { return OutE[i].W; }
00266
```

4.63.2.19 template<class WT> int sdVertex< WT >::HasEdgeFrom (const sdVertex< WT > & *from*) const [inline]

Definition at line 240 of file sdvertex.inc.

Referenced by Connected().

```
00241   { return (GetInEdgeIndex(from.ID) != -1); }
00242
```

4.63.2.20 template<class WT> int sdVertex< WT >::HasEdgeTo (const sdVertex< WT > & *to*) const [inline]

Definition at line 236 of file sdvertex.inc.

Referenced by Connected(), and sdGraph::HasEdge().

```
00237   { return (GetOutEdgeIndex(to.ID) != -1); }
00238
```

4.63.2.21 template<class WT> int sdVertex< WT >::InDegree () const [inline]

Definition at line 89 of file sdvertex.inc.

Referenced by Degree().

```
00090   { return NumI; }
00091
```

4.63.2.22 template<class WT = double> sdVertex<WT>::MAKECOMPARISONS_WITH_COMPARE (sdVertex< WT >)

4.63.2.23 template<class WT> int sdVertex< WT >::OutDegree () const [inline]

Definition at line 85 of file sdvertex.inc.

Referenced by Degree().

```
00086   { return NumO; }
00087
```

4.63.2.24 template<class WT> void sdVertex< WT >::PrintEdges (ostream & *os*) const [inline]

Definition at line 118 of file sdvertex.inc.

```
00119   { PrintOutEdges(os); PrintInEdges(os); }
00120
```

4.63.2.25 template<class WT> void sdVertex< WT >::PrintInEdges (ostream & *os*) const [inline]

Definition at line 111 of file sdvertex.inc.

Referenced by PrintEdges().

```
00112   {
00113     for(int i=0; i<NumI; ++i)
00114         os << '\t' << InE[i].V->ID << '\t' << ID << '\t' << InE[i].W << '\n';
00115
```

4.63.2.26 template<class WT> void sdVertex< WT >::PrintOutEdges (ostream & *os*) const [inline]

Definition at line 104 of file sdvertex.inc.

Referenced by sdGraph::Output(), and PrintEdges().

```
00105   {
00106     for(int i=0; i<NumO; ++i)
00107         os << '\t' << ID << '\t' << OutE[i].V->ID << '\t' << OutE[i].W << '\n';
00108
```

4.63.2.27 template<class WT> void sdVertex< WT >::SortEdges () [inline]

Definition at line 97 of file sdvertex.inc.

Referenced by sdGraph::Copy().

```
00098   {
00099     QuickSort(OutE, NumO);
00100     QuickSort(InE, NumI);
00101
```

4.63.2.28 template<class WT> sdVertex< WT > & sdVertex< WT >::operator() (int *i*) [inline]

Definition at line 202 of file sdvertex.inc.

```
00203   { return GetInNeighbor(i); }
00204
```

4.63.2.29 template<class WT> const sdVertex< WT > & sdVertex< WT >::operator() (int *i*) const [inline]

Definition at line 198 of file sdvertex.inc.

```
00199   { return GetInNeighbor(i); }
00200
```

4.63.2.30 template<class WT> const sdVertex< WT > & sdVertex< WT >::operator= (const sdVertex< WT > & *sdv*) [inline]

Definition at line 126 of file sdvertex.inc.

```
00127   {
00128     sdEdge<WT> * OutE_tmp = OutE, * InE_tmp = InE;
00129
00130     if(this != &sdv)
00131       {
00132         if(sdv.NumO<1)
00133             OutE = 0;
00134         else if(!(OutE = new sdEdge<WT>[sdv.NumO]))
00135           {
00136             OutE = OutE_tmp;
00137             return *this;
00138           }
00139         if(sdv.NumI<1)
00140             InE = 0;
00141         else if(!(InE  = new sdEdge<WT>[sdv.NumI]))
00142           {
00143             delete[] OutE;
00144             OutE = OutE_tmp;
00145             InE  = InE_tmp;
00146             return *this;
00147           }
00148         delete[] OutE_tmp;
00149         delete[] InE_tmp;
00150
00151         ID   = sdv.ID;
00152         NumO = sdv.NumO;
00153         NumI = sdv.NumI;
00154
00155         for(int i=0; i<NumO; ++i)
00156           {
00157             OutE[i].V = sdv.OutE[i].V;
00158             OutE[i].W = sdv.OutE[i].W;
00159           }
00160         for(int i=0; i<NumI; ++i)
00161           {
00162             InE[i].V = sdv.InE[i].V;
00163             InE[i].W = sdv.InE[i].W;
00164           }
00165       }
00166
```

4.63.2.31 template<class WT> sdVertex< WT > & sdVertex< WT >::operator[] (int *i*) [inline]

Definition at line 186 of file sdvertex.inc.

```
00187   { return GetOutNeighbor(i); }
00188
```

4.63.2.32 template<class WT> const sdVertex< WT > & sdVertex< WT >::operator[] (int *i*) const [inline]

Definition at line 182 of file sdvertex.inc.

```
00183   { return GetOutNeighbor(i); }
00184
```

4.63.3 Friends And Related Function Documentation

4.63.3.1 template<class WT = double> int Connected (const sdVertex< WT > & *v1*, const sdVertex< WT > & *v2*) [friend]

Definition at line 244 of file sdvertex.inc.

```
00245   { return (v1.HasEdgeTo(v2) || v1.HasEdgeFrom(v2)); }
00246
```

4.63.3.2 template<class WT = double> ostream& operator (ostream & *os*, const sdVertex< WT > & *vert*) [friend]

4.63.4 Member Data Documentation

4.63.4.1 template<class WT = double> int sdVertex<WT>::ID

Definition at line 52 of file sdvertex.h.

4.63.4.2 template<class WT = double> sdEdge< WT > * sdVertex<WT>::InE

Definition at line 60 of file sdvertex.h.

4.63.4.3 template<class WT = double> int sdVertex<WT>::NumI

Definition at line 56 of file sdvertex.h.

4.63.4.4 template<class WT = double> int sdVertex<WT>::NumO

Definition at line 55 of file sdvertex.h.

4.63.4.5 template<class WT = double> sdEdge< WT > * sdVertex<WT>::OutE

Definition at line 59 of file sdvertex.h.

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

- **sdvertex.h**
- **sdvertex.inc**

4.64 Segment Class Reference

Public Methods

- **Segment** ()
- **Segment** (double a, double b)
- **~Segment** ()
- const Segment& **operator=** (const Segment &s)

Public Attributes

- double **start**
- double **end**

Friends

- int **operator>** (const Segment &s1, const Segment &s2)
- int **operator<** (const Segment &s1, const Segment &s2)
- ostream& **operator<<** (ostream &os, const Segment &seg)

4.64.1 Constructor & Destructor Documentation

4.64.1.1 Segment::Segment () [inline]

Definition at line 23 of file segments.inc.

```
00023           {}
00024
```

4.64.1.2 Segment::Segment (double *a*, double *b*) [inline]

Definition at line 25 of file segments.inc.

```
00025                               {
00026     if (a<b){
00027       start = a;
00028       end = b;
00029     } else {
00030       start = b;
00031       end = a;
00032     }
00033
```

4.64.1.3 Segment::~Segment () [inline]

Definition at line 35 of file segments.inc.

```
00036     {
00037
```

4.64.2 Member Function Documentation

4.64.2.1 const Segment & Segment::operator= (const Segment & *s*) [inline]

Definition at line 39 of file segments.inc.

```
00040     {
00041       if (this != &s)
00042       {
00043         start = s.start;
00044         end = s.end;
00045         //      width = s.width;
00046       }
00047
```

4.64.3 Friends And Related Function Documentation

4.64.3.1 int operator< (const Segment & *s1*, const Segment & *s2*) [friend]

Definition at line 62 of file segments.inc.

```
00063     {
00064       // order on START value
00065       if(s1.start < s2.start) return 1;
00066       else if(s1.start > s2.start) return 0;
00067       // If the Segment have the same start position;
00068       if(s1.end < s2.end) return 1;
00069       else if(s1.end > s2.end) return 0;
00070       // If the Segment are the same
00071       return 0;
00072
```

4.64.3.2 ostream & operator<< (ostream & *os*, const Segment & *seg*) [friend]

Definition at line 74 of file segments.inc.

```
00075     {
00076       os << seg.start << " " << seg.end;// << " " << seg.width;
00077       return os;
00078
```

4.64.3.3 int operator> (const Segment & *s1*, const Segment & *s2*) [friend]

Definition at line 50 of file segments.inc.

```
00051     {
00052       // order on START value
00053       if(s1.start > s2.start) return 1;
00054       else if(s1.start < s2.start) return 0;
00055       // If the Segment have the same start position;
00056       if(s1.end > s2.end) return 1;
00057       else if(s1.end < s2.end) return 0;
00058       // If the Segment are the same
00059       return 0;
00060
```

4.64.4 Member Data Documentation

4.64.4.1 double Segment::end

Definition at line 81 of file segments.inc.

4.64.4.2 double Segment::start

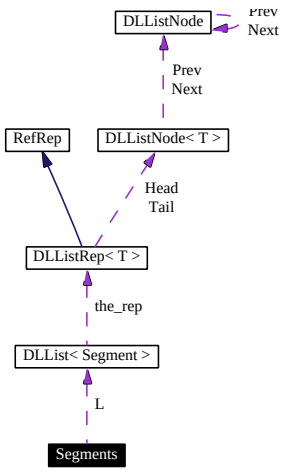
Definition at line 80 of file segments.inc.

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

- **segments.inc**

4.65 Segments Class Reference

Collaboration diagram for Segments:



Public Methods

- **Segments** ()
- **~Segments** ()
- double **GetNum** (void)
- **Segment GetSeg** (int j)
- void **Add** (double start, double end)
- void **Add** (**Segment** seg)
- void **QSortReduce** ()
- **Segment QSortReduceViol** (double testval)
- **Segment Pop** (void)
- int **Number** (void)
- void **Clear** ()

Private Methods

- bool **Overlap** (**Segment** &s1, **Segment** &s2)
- **Segment Join** (**Segment** &s1, **Segment** &s2)

Private Attributes

- **DLList L**

4.65.1 Constructor & Destructor Documentation

4.65.1.1 Segments::Segments () [inline]

Definition at line 93 of file segments.inc.

```
00093               { }
00094
```

4.65.1.2 Segments::~Segments () [inline]

Definition at line 94 of file segments.inc.

```
00094                { }
00095
```

4.65.2 Member Function Documentation

4.65.2.1 void Segments::Add (Segment *seg*) [inline]

Definition at line 109 of file segments.inc.

```
00109                        {
00110     L.AddToFront(seg);      // unsorted Segment list
00111
```

4.65.2.2 void Segments::Add (double *start*, double *end*) [inline]

Definition at line 104 of file segments.inc.
Referenced by gwNumerics::addLPERTSegments(), and gwNumerics::addVDWSegments().

```
00104                                    {
00105     Segment s(start,end);
00106     L.AddToFront(s);      // unsorted Segment list
00107
```

4.65.2.3 void Segments::Clear () [inline]

Definition at line 197 of file segments.inc.

```
00198     {L.Clear();}
00199
```

4.65.2.4 double Segments::GetNum (void) [inline]

Definition at line 96 of file segments.inc.
Referenced by gwNumerics::SetHCPosition2(), and gwNumerics::addLPERTSegments().

```
00096                       {
00097     return L.Number();
00098
```

4.65.2.5 Segment Segments::GetSeg (int *j*) [inline]

Definition at line 100 of file segments.inc.
Referenced by gwNumerics::SetHCPosition2(), and gwNumerics::addLPERTSegments().

```
00100                         {
00101     return L.Get(j);
00102
```

4.65.2.6 Segment Segments::Join (Segment & *s1*, Segment & *s2*) [inline, private]

Definition at line 211 of file segments.inc.
Referenced by QSortReduce().

```
00212     {
00213
00214       Segment returnSegment;
00215       returnSegment.start = (s1.start>s2.start) ? s2.start : s1.start;
00216       returnSegment.end = (s1.end<s2.end) ? s2.end : s1.end;
00217       //      returnSegment.width = returnSegment.end - returnSegment.start;
00218       return returnSegment;
00219
```

4.65.2.7 int Segments::Number (void) [inline]

Definition at line 194 of file segments.inc.

```
00195     {return(L.Number());}
00196
```

4.65.2.8 bool Segments::Overlap (Segment & *s1*, Segment & *s2*) [inline, private]

Definition at line 202 of file segments.inc.
Referenced by QSortReduce().

```
00203     {
00204       if ((s1.start >= s2.start)&&(s1.start <= s2.end)) return true;
00205       if ((s2.start >= s1.start)&&(s2.start <= s1.end)) return true;
00206       if ((s1.end >= s2.start)&&(s1.end <= s2.end)) return true;
00207       if ((s2.end >= s1.start)&&(s2.end <= s1.end)) return true;
00208       return false;
00209
```

4.65.2.9 Segment Segments::Pop (void) [inline]

Definition at line 191 of file segments.inc.

```
00192     {return(L.RemoveFromFront());}
00193
```

4.65.2.10 void Segments::QSortReduce () [inline]

Definition at line 113 of file segments.inc.
Referenced by QSortReduceViol().

```
00113                     {
00114     int num = L.Number();
00115     if (num == 0) return;
00116     // Sort the list using another array
00117     Segment *segarray = new Segment[num];
```

```
00118     for (int i= 0;i<num;i++)
00119       segarray[i] = L.RemoveFromFront(); // Create array for quicksort
00120     RecQuickSort(segarray, 0, num-1);  // Sort it (from search-sort.h)
00121
00122     L.Clear(); // sanity ck debug
00123
00124     // Reduce as they are added back to the list.
00125     Segment TempSeg = segarray[0];
00126     int j = 1;
00127     // the next 9 lines are a gem. --jd
00128     while (j < num){
00129       if(Overlap(TempSeg,segarray[j])){
00130         TempSeg = Join(TempSeg,segarray[j++]);
00131       } else {
00132         L.AddToBack(TempSeg);
00133         TempSeg = segarray[j++];
00134       }
00135     }
00136
```

4.65.2.11 Segment Segments::QSortReduceViol (double *testval*) [inline]

Definition at line 142 of file segments.inc.

Referenced by gwNumerics::SetHCPosition2().

```
00142                                       {
00143     // Returns segment that overlaps the testval.  This check REQUIES
00144     // that the seglist be sorted and reduced for efficient checking,
00145     // so this function begins with a call to QSortReduce.
00146
00147     Segment testseg(testval,testval);
00148
00149     QSortReduce();
00150
00151 #ifdef JOB_VERBOSE2
00152     int num = L.Number();
00153     cerr << "final number segments = "<< num << "; "      ;
00154     for (int i=0;i<num;i++) cerr << "(" << L.Get(i) << ") " ;
00155     cerr << endl;
00156 #endif
00157
00158     if (L.Number()>0){
00159       int segind = 0;
00160       testseg = L.Get(segind++);
00161 #ifdef JOB_VERBOSE2
00162       cerr << "walking up through segment: ("<< testseg <<") ;?test overlap "<<testval<< endl;
00163 #endif
00164       while (testseg.end < testval){
00165         if (segind >= L.Number()) break;
00166         testseg = L.Get(segind++);
00167       }
00168       // test for violation
00169       if ((testseg.start < testval) && (testseg.end > testval)){
00170 #ifdef JOB_VERBOSE2
00171     cerr << "Violating segment: ("<< testseg <<")"<< endl;
00172 #endif
00173
00174         return testseg;
00175       }
00176       testseg.start = 0; // returns zero segment if ok
00177       testseg.end = 0;
00178     } else {
```

```
00179       // no segments at all in this class
00180       testseg.start = 0; // returns zero segment if ok
00181       testseg.end = 0;
00182     }
00183
```

4.65.3 Member Data Documentation

4.65.3.1 DLList< Segment > Segments::L [private]

Definition at line 89 of file segments.inc.

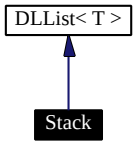
The documentation for this class was generated from the following file:

- **segments.inc**

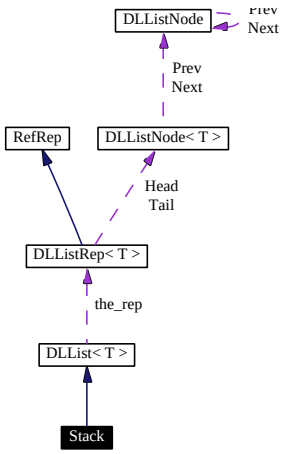
4.66 Stack Class Template Reference

`#include <stack.h>`

Inheritance diagram for Stack



Collaboration diagram for Stack:



Public Methods

- void **Push** (const T &data)
- void **Push** (const Stack< T > &stk)
- T **Pop** ()
- const T& **Peek** () const
- const Stack<T>& **operator+=** (const Stack< T > &stk)

Friends

- Stack<T> **operator+** (Stack< T > x, const Stack< T > &y)

template<class T> class Stack

4.66.1 Member Function Documentation

4.66.1.1 template<class T> const T & Stack<T>::Peek () const [inline]

Definition at line 22 of file stack.h.
Referenced by IdGen::Peek().

```
00022                                   { return Get(0); }
00023
```

4.66.1.2 template<class T> T Stack<T>::Pop () [inline]

Definition at line 21 of file stack.h.

Referenced by IdGen::Get(), and df_iterator::operator++().

```
00021                       { return RemoveFromFront(); }
00022
```

4.66.1.3 template<class T> void Stack<T>::Push (const Stack< T > & *stk*) [inline]

Definition at line 20 of file stack.h.

```
00020                                           { AddToFront(stk); }
00021
```

4.66.1.4 template<class T> void Stack<T>::Push (const T & *data*) [inline]

Definition at line 19 of file stack.h.

Referenced by IdGen::Free(), df_iterator::operator++(), and operator+=().

```
00019                                       { AddToFront(data); }
00020
```

4.66.1.5 template<class T> const Stack< T > & Stack<T>::operator+=<T> (const Stack< T > & *stk*) [inline]

Definition at line 23 of file stack.h.

```
00024                 { Push(stk); return *this; }
00025
```

4.66.2 Friends And Related Function Documentation

4.66.2.1 template<class T> Stack< T > operator+<T> (Stack< T > *x*, const Stack< T > & *y*) [friend]

Definition at line 26 of file stack.h.

```
00027                 { return x+=y; }
00028
```

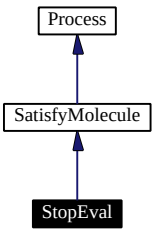
The documentation for this class was generated from the following file:

- **stack.h**

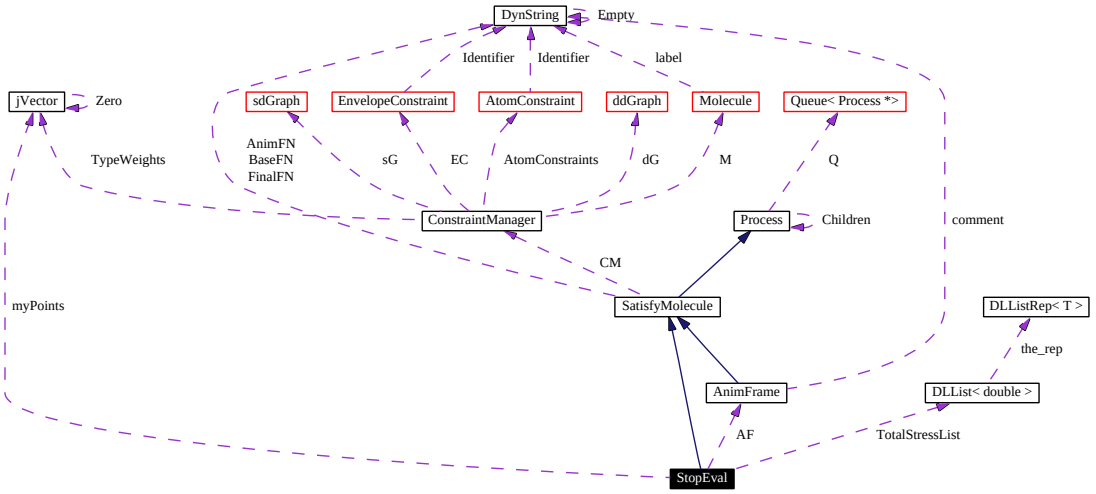
4.67 StopEval Class Reference

`#include <jd_stop_eval.h>`

Inheritance diagram for StopEval



Collaboration diagram for StopEval:



Public Methods

- **StopEval** (const StopEval &ca)
- **StopEval** (**ConstraintManager** &**CM**)
- **~StopEval** ()
- int **run** ()
- int **check_stress_max_tot** (double, double)
- int **check_low_error** (double, double)
- void **cycle_header** (void)
- void **writebest** (void)

Static Public Attributes

- bool **Enable** = true
- bool **SolveOnly** = false
- bool **Rtp** = false

  Real time plotting output - single double outputs for plotting.

- bool **Quiet** = false

  run-time ajdustable flag to stop default output.

- bool **VerboseOn** = false

  run-time ajdustable flag to provide verbose output.

Private Attributes

- **AnimFrame*** **AF**
- ofstream* **of**
- **jVector*** **myPoints**
- double **LowErr**
- double **BestAveErr**
- double **BestMaxErr**

Static Private Attributes

- int **LookBack** = 50

  number of calls to StopEval to remember.

- **DLList**<double> **TotalStressList**
- double **MinTotalStress**
- double **MinofMaxStress**

  the overall min of the max on each list of stress calls on an iteration.

- double **MinofMaxAssocTotal**
- double **MinLookBackStress**
- double **MaxLookBackStress**
- double **MeanLookBackTotalStress**
- double **SumXX**

  Variables for linear regression these are maintained over the lookback calls to **StopEval::run***()* (p. 493).

- double **SumXY**
- double **SumX**
- double **SumY**
- double **SumYY**
- double **Variance**
- int **NumCalls** = 0
- int **NumLprAdj** = 0
- int **NumLprLim** = 10
- int **LastLprAdj** = 0
- int **BestIter** = 0
- double **LookBackSlope**
- double **LookBackD**

4.67.1 Detailed Description

//

File: **jd_stop_eval.h**

Created by: Jon Dugan

Evaluates the molecule state to determine iteration exit conditions.

//

Definition at line 28 of file jd_stop_eval.h.

4.67.2 Constructor & Destructor Documentation

4.67.2.1 StopEval::StopEval (const StopEval & *se*)

Definition at line 35 of file jd_stop_eval.C.

```
00036   : SatisfyMolecule(*(se.CM))
00037 {
00038   TotalStressList = se.TotalStressList;
00039   DIE(" Copy constructor for stop eval called ");
00040
00041
```

4.67.2.2 StopEval::StopEval (ConstraintManager & *cm_in*)

Definition at line 43 of file jd_stop_eval.C.

```
00044   : SatisfyMolecule(cm_in)
00045 {
00046
00047   AF = new AnimFrame(cm_in);
00048   of = 0;
00049   myPoints = new jVector [CM->numAtoms()];
00050   LowErr = INF;
00051   BestAveErr= INF;
00052   BestMaxErr = INF;
00053
```

4.67.2.3 StopEval::~StopEval ()

Definition at line 55 of file jd_stop_eval.C.

```
00056 {
00057   delete AF; AF = 0;
00058   if (of) delete of; of = 0;
00059   delete [] myPoints;
00060
```

4.67.3 Member Function Documentation

4.67.3.1 int StopEval::check_low_error (double *max_err*, double *ave_err*)

Definition at line 62 of file jd_stop_eval.C.

Referenced by AtomBuildup::run().

```
00062                                                          {
00063   double err = 0.1 * ave_err + max_err;
00064   if (err < LowErr) {
00065     LowErr = err;
00066     BestIter = NumCalls;
00067     for (int i= 0; i < CM->numAtoms(); i ++){
00068       myPoints[i] = CM->getMolecule().getPosition(i); // save best positions
00069     }
00070     BestAveErr = ave_err;
00071     BestMaxErr = max_err;
00072     return 1;
00073   }
00074
```

4.67.3.2 int StopEval::check_stress_max_tot (double *max_test*, double *total_test*)

Definition at line 77 of file jd_stop_eval.C.

Referenced by AtomBuildup::run().

```
00078 {  // if either passes, stop
00079
00080   double max = -1*INF;
00081   double tot = 0;
00082   jVector currStressList = CM->getStressList();
00083
00084   for (int i =0; i <  currStressList.Length(); i++){
00085     if (max < currStressList[i]) max = currStressList[i];
00086     tot += currStressList[i];
00087   }
00088
```

4.67.3.3 void StopEval::cycle_header (void)

Definition at line 101 of file jd_stop_eval.C.

```
00101                              {
00102   cerr << "\nCycle   Avg Err    Max Err  Tot Err  %pert  Mean Total       Slope  Variance"
00103        << LookBack << "\n"
00104        << "-----   -------    -------  -------   ----  ----------       -----  -----------      "
00105        << endl;
00106
00107
```

4.67.3.4 int StopEval::run (void) [virtual]

Execute this process. Returns the exit status of the process (0 is success, non-zero is error).

Reimplemented from **Process** (p. 396).

Definition at line 109 of file jd_stop_eval.C.

```
00109                  {
00110 #ifdef JOB_VERBOSE
00111   cerr << "DIDJOB:\tStopEval::run()" << endl;
00112 #endif
```

```
00113   /*
00114
00115 //
00116 //   // initialize max, mean and total stress
00117 //   double maxStress = 0;
00118 //   double totalStress = 0;
00119 //   jVector currStressList = CM->getStressList();
00120 //
00121 //   for (int i =0; i <  currStressList.Length(); i++){
00122 //     if (maxStress < currStressList[i]) maxStress = currStressList[i];
00123 //     totalStress += currStressList[i];
00124 //   }
00125 //   double meanStress = totalStress / currStressList.Length();
00126 //
00127 //   double PercentLP = gwNumerics::NumLP/(double)(gwNumerics::NumLP + gwNumerics::NumNotLP);
00128 //   //  cerr << endl <<", " << gwNumerics::get_NumLP() << ", " << gwNumerics::get_NumNotLP();
00129 //   gwNumerics::reset_LP_Count(); //reset
00130 //
00131 //   NumCalls++;
00132 //
00133 //   if (!Quiet) {
00134 //     if (NumCalls == 1){cycle_header();}
00135 //     // ref see
00136 //     // http://gcc.gnu.org/onlinedocs/iostream_3.html
00137 //     //
00138 //     cerr << endl ;
00139 //     cerr << setw(4) << setiosflags(ios::left) << resetiosflags(ios::showpoint)
00140 //       << NumCalls ;
00141 //
00142 //     cerr << resetiosflags(ios::left) << setiosflags(ios::showpoint)
00143 //       << setw(11)  << setprecision(6)
00144 //       << meanStress
00145 //
00146 //       << setw(11)  << setprecision(6)
00147 //       << maxStress
00148 //
00149 //       << setw(9)  << setprecision(6)
00150 //       << totalStress
00151 //
00152 //       << " "
00153 //       << setw(6) << setprecision(2) << resetiosflags(ios::showpoint)
00154 //       << 100*PercentLP
00155 //       << setprecision(4);
00156 //   }
00157 //
00158 //   if (Rtp) {
00159 //     cout << totalStress << " " << endl;
00160 //   }
00161 //
00162 //   // disable other stuff
00163 //   //  return 0;
00164 //
00165 //
00166 //   if (!Enable) return 0; //keep running
00167 //   if (NumCalls < 5) return 0; //never quit before first 5 calls
00168 //   // FIX  magix number == 5, make a static, a default val above and add to interface
00169 //
00170 // #ifdef JOB_VERBOSE2
00171 //   cerr << "Value: \tStopEval::run() Max. Stress Atom Value(EvalCall) = " \
00172 //        << maxStress << " ("<<NumCalls<<")"<< endl;
00173 //   cerr << "Value: \tStopEval::run() Total Stress Value = "<< totalStress << endl;
00174 // #endif
00175 //
00176 //   if (MinTotalStress > totalStress) MinTotalStress = totalStress;  // maintain mins
00177 //   if (MinofMaxStress > maxStress) {
```

```
00178 //     MinofMaxStress = maxStress ;
00179 //     MinofMaxAssocTotal = totalStress;
00180 //     BestIter = NumCalls;
00181 //     for (int i= 0; i < CM->numAtoms(); i ++){
00182 //       myPoints[i] = CM->getMolecule().getPosition(i); // save best positions
00183 //     }
00184 //   }
00185 //
00186 //   TotalStressList.AddToFront(totalStress);
00187 //
00188 //   int listlength = TotalStressList.Number();    //Must check this, may not eval every cycle.
00189 //
00190 //   if (listlength == LookBack) { //set up linear regression vars once
00191 //
00192 //     double tempX, tempY;
00193 //     for (int i = 0; i < listlength; i++){
00194 //       tempY = TotalStressList.Get(i);
00195 //       tempX = (double) NumCalls - i;
00196 //       MeanLookBackTotalStress += tempY/LookBack;
00197 //       SumXX += tempX*tempX;
00198 //       SumXY += tempX*tempY;
00199 //       SumX  += tempX;
00200 //       SumY  += tempY;
00201 //       SumYY += tempY*tempY;
00202 //
00203 //       if (MinLookBackStress > tempY) MinLookBackStress = tempY;    // set init min/max over list
00204 //       else if (MaxLookBackStress < tempY) MaxLookBackStress = tempY;
00205 //
00206 //     } // done lookback times
00207 //
00208 //     // now set init variance
00209 //     Variance = ((LookBack*SumYY)-(SumY*SumY))/(LookBack*(LookBack-1));
00210 //
00211 //   } // done once to setup lookback vars
00212 //
00213 //   if (listlength > LookBack) { //ie listlength == lookback+1
00214 //     // calcualte slope and mean over lookback length.  Currently I
00215 //     // will be simply adding data to the end and removing it from the
00216 //     // back to maintain the required data for the linear regression.
00217 //     double OldYVal = TotalStressList.RemoveFromBack();
00218 //     double OldX = (double) (NumCalls - LookBack);
00219 //     double OldSumYY = SumYY;
00220 //     double OldSumY = SumY;
00221 //     if (VerboseOn){
00222 //       cerr <<"StopEval:\t" ;
00223 //       cerr << "Num: " << NumCalls <<"  \t";
00224 //       cerr << "Added val: " << totalStress <<"  \t";
00225 //       cerr << "Removed val: " << OldYVal << endl;
00226 //     }
00227 //
00228 //     // handle min/max over list
00229 //     if (MinLookBackStress > totalStress) MinLookBackStress = totalStress;
00230 //     else if (MaxLookBackStress < totalStress) MaxLookBackStress = totalStress;
00231 //     else {
00232 //       if (MinLookBackStress == OldYVal) {
00233 //      DLList<double>::ConstIterator it;
00234 //      for(it = TotalStressList.GetConstIterator(); it; ++it)
00235 //        if (MinLookBackStress > (*it)) MinLookBackStress = (*it);
00236 //       }
00237 //       else if (MaxLookBackStress == OldYVal){
00238 //      DLList<double>::ConstIterator it;
00239 //      for(it = TotalStressList.GetConstIterator(); it; ++it)
00240 //        if (MaxLookBackStress < (*it)) MaxLookBackStress = (*it);
00241 //       }
00242 //     }
```

```
00243 //
00244 //
00245 //     MeanLookBackTotalStress = MeanLookBackTotalStress + ((totalStress-OldYVal)/LookBack);
00246 //
00247 //     SumXX = SumXX + ((NumCalls)*(NumCalls) - OldX*OldX); // actually NumCalls-OldX=LookBack-1
00248 //     SumXY = SumXY + (NumCalls*totalStress - OldX*OldYVal);
00249 //     SumX  = SumX + (double) LookBack;
00250 //     SumY  = SumY + (totalStress - OldYVal);
00251 //     SumYY  = SumYY + (totalStress*totalStress-OldYVal*OldYVal);
00252 //
00253 //     LookBackD = LookBack*SumXX - SumX*SumX;
00254 //     LookBackSlope = (LookBack* SumXY-SumX*SumY)/LookBackD;
00255 //
00256 //     Variance += (((LookBack*SumYY)-(SumY*SumY))-((LookBack*OldSumYY)-(OldSumY*OldSumY)))/(LookBack*(LookBack-1));
00257 //     //    cerr << NumCalls << " " << totalSat << " " << Variance << endl;
00258 //
00259 //     if (VerboseOn){
00260 //       cerr <<"StopEval:\t" ;
00261 //       cerr << "Call Number: " << NumCalls <<"\t";
00262 //       cerr << "OverallMin: " << MinTotalStress << "\t";
00263 //       cerr << "MaxAtomStress: " << maxStress << "\t";
00264 //       cerr << "Mean: " << MeanLookBackTotalStress << "\t";
00265 //       cerr << "Slope: " << LookBackSlope << "\t";
00266 //       cerr << "Variance: " << Variance << endl;
00267 //     }
00268 //
00269 //     if (!Quiet){
00270 //
00271 //       cerr << setw(11)  << setiosflags(ios::showpoint)
00272 //         << MeanLookBackTotalStress
00273 //
00274 //         << setw(11)
00275 //         << LookBackSlope
00276 //
00277 //         << setw(10)
00278 //         << Variance;
00279 //
00280 //     }
00281 //
00282 //     // CASE FOR SOLVING CONSTRAINTS ON ALL ATOMS
00283 //     if (maxStress < 0.001 && Variance < 5)  {  //MAGIC NUMBERS!!
00284 //       cerr << "\nSTOP by case 1 (solved)";
00285 //       writebest();
00286 //       return 1; //stop now
00287 //     }
00288 //
00289 //     if (SolveOnly) return 0; //keep running, don't check further
00290 //
00291 //
00292 //     //
00293 //     // Simulated anealing type stuff with lpertfactor going up and down.
00294 //     //
00295 //
00296 // //     if ((Variance < 5) && (LookBackSlope > -0.01) &&
00297 // //   (NumCalls - LastLprAdj > LookBack) && (maxStress > 0.1))
00298 // //
00299 // //       {
00300 // //   gwNumerics::Mul_LPertFacor(1.36);
00301 // //
00302 // //   NumLprAdj++;
00303 // //   cerr << "\nLPR ADJUST ++  case 2a (stalled, no variance) cycle: " << NumCalls
00304 // //        << "# " << NumLprAdj << endl;
00305 // //
00306 // //   LastLprAdj = NumCalls;
00307 // //
```

```
00308 // //   if (NumLprAdj == NumLprLim) {
00309 // //     cerr << "\nLPR RESET =0.5  case 2a (stalled, not improving) cycle: " << NumCalls
00310 // //          << "# " << NumLprAdj << endl;
00311 // //     gwNumerics::Set_LPertFacor(1);
00312 // //     CM->RandomPerturb(20);
00313 // //     NumLprAdj = 0;
00314 // //   }
00315 // //   if (NumLprAdj > NumLprLim + 5) {
00316 // //     cerr << "\nSTOP ++  case 2a (stalled, no variance) cycle: " << NumCalls
00317 // //          << "# " << NumLprAdj << endl;
00318 // //     writebest();
00319 // //     return 1; // STOP
00320 // //
00321 // //       }
00322 // //       return 0;
00323 // //     }
00324 //
00325 //
00326 // //     if ((
00327 // //    (Variance > 5000) && (LookBackSlope > 8) &&
00328 // //    (NumCalls - LastLprAdj >  LookBack) && (PercentLP > .2)
00329 // //    ) || (
00330 // //          (NumCalls - BestIter > 3 * LookBack)
00331 // //          && (NumCalls - LastLprAdj >  LookBack )
00332 // //         )
00333 // //   ) {
00334 // //       gwNumerics::Mul_LPertFacor(.81);
00335 // //       NumLprAdj--;
00336 // //
00337 // //       cerr << "\nLPR ADJUST --  case 2b (stalled, not improving) cycle: " << NumCalls
00338 // //      << "# " << NumLprAdj << endl;
00339 // //
00340 // //       LastLprAdj = NumCalls;
00341 // //
00342 // //       if (NumLprAdj == (-1)*NumLprLim) {
00343 // //    cerr << "\nLPR RESET =3  case 2b (stalled, not improving) cycle: " << NumCalls
00344 // //         << "# " << NumLprAdj << endl;
00345 // //    gwNumerics::Set_LPertFacor(3);
00346 // //    CM->RandomPerturb(20);
00347 // //    NumLprAdj = 0;
00348 // //       }
00349 // //
00350 // //       if (NumLprAdj < (-1)*NumLprLim - 7) {
00351 // //    cerr << "\nSTOP --  case 2b (stalled, not improving) cycle: " << NumCalls
00352 // //         << "# " << NumLprAdj << endl;
00353 // //    writebest();
00354 // //    return 1; // STOP
00355 // //       }
00356 // //       return 0;
00357 // //     }
00358 //
00359 //
00360 //
00361 //
00362 //     // getting worse
00363 //     if ((LookBackSlope > 100) && (NumCalls - LastLprAdj >  LookBack ) &&
00364 //      (MinTotalStress < MeanLookBackTotalStress/ 5) &&
00365 //      (NumLprAdj < 1)) {
00366 //       cerr << "\nSTOP by case 3 (getting worse) Slope = " << LookBackSlope << endl;
00367 //       writebest();
00368 //       return 1;
00369 //     }
00370 //
00371 //
00372 //   }
```

```
00373
```

4.67.3.5 void StopEval::writebest (void)

Definition at line 379 of file jd_stop_eval.C.
Referenced by AtomBuildup::run().

```
00379                              {
00380
00381   int N = CM->numAtoms();
00382
00383   DynString comment;
00384   DynString FN;
00385   comment = BaseFN +
00386     DynString(" ;Best Iteration ") + DynString(BestIter) +
00387     DynString(" ;Ave Err = ") + DynString (BestAveErr) +
00388     DynString(" ;Max Err = ") + DynString (BestMaxErr);
00389   FN = BaseFN + DynString(".best.xmol");
00390
00391   char * s = &(FN[0]);  // ACK  not good, but it doesn't leak. -jd
00392   ofstream os(s);
00393
00394   os << N << "\n";
00395   os << comment << "\n";
00396   for(int i=0; i<N; ++i)
00397     os << CM->getAtom(i).Type() <<" "<< myPoints[i] << endl;
00398
00399
00400
```

4.67.4 Member Data Documentation

4.67.4.1 AnimFrame * StopEval::AF [private]

Definition at line 79 of file jd_stop_eval.h.

4.67.4.2 double StopEval::BestAveErr [private]

Definition at line 139 of file jd_stop_eval.h.

4.67.4.3 int StopEval::BestIter = 0 [static, private]

Definition at line 31 of file jd_stop_eval.C.

4.67.4.4 double StopEval::BestMaxErr [private]

Definition at line 140 of file jd_stop_eval.h.

4.67.4.5 bool StopEval::Enable = true [static]

Definition at line 69 of file globals.C.

4.67.4.6 int StopEval::LastLprAdj = 0 [static, private]

Definition at line 30 of file jd_stop_eval.C.

4.67.4.7 int StopEval::LookBack = 50 [static, private]

number of calls to StopEval to remember.

Definition at line 68 of file globals.C.

4.67.4.8 double StopEval::LookBackD [static, private]

Definition at line 136 of file jd_stop_eval.h.

4.67.4.9 double StopEval::LookBackSlope [static, private]

Definition at line 135 of file jd_stop_eval.h.

4.67.4.10 double StopEval::LowErr [private]

Definition at line 138 of file jd_stop_eval.h.

4.67.4.11 double StopEval::MaxLookBackStress [static, private]

Definition at line 111 of file jd_stop_eval.h.

4.67.4.12 double StopEval::MeanLookBackTotalStress [static, private]

mean of the lookback list vals on total stress, incrementally updated after lookback number of calls to stopeval

Definition at line 117 of file jd_stop_eval.h.

4.67.4.13 double StopEval::MinLookBackStress [static, private]

The Minimum (max) total Stress value observed over lookback calls to stop_eval.

Definition at line 110 of file jd_stop_eval.h.

4.67.4.14 double StopEval::MinTotalStress [static, private]

Definition at line 93 of file jd_stop_eval.h.

4.67.4.15 double StopEval::MinofMaxAssocTotal [static, private]

the asociated total stress on the saved "best" molecule defined by MinofMaxStress'

Definition at line 104 of file jd_stop_eval.h.

4.67.4.16 double StopEval::MinofMaxStress [static, private]

the overall min of the max on each list of stress calls on an iteration.

Definition at line 98 of file jd_stop_eval.h.

4.67.4.17 int StopEval::NumCalls = 0 [static, private]

Definition at line 27 of file jd_stop_eval.C.

4.67.4.18 int StopEval::NumLprAdj = 0 [static, private]

Definition at line 28 of file jd_stop_eval.C.

4.67.4.19 int StopEval::NumLprLim = 10 [static, private]

Definition at line 29 of file jd_stop_eval.C.

4.67.4.20 bool StopEval::Quiet = false [static]

run-time ajdustable flag to stop default output.

Definition at line 72 of file globals.C.

4.67.4.21 bool StopEval::Rtp = false [static]

Real time plotting output – single double outputs for plotting.

Definition at line 73 of file globals.C.

4.67.4.22 bool StopEval::SolveOnly = false [static]

Definition at line 70 of file globals.C.

4.67.4.23 double StopEval::SumX [static, private]

Definition at line 123 of file jd_stop_eval.h.

4.67.4.24 double StopEval::SumXX [static, private]

Variables for linear regression these are maintained over the lookback calls to **StopEval::run**() (p. 493).

Definition at line 121 of file jd_stop_eval.h.

4.67.4.25 double StopEval::SumXY [static, private]

Definition at line 122 of file jd_stop_eval.h.

4.67.4.26 double StopEval::SumY [static, private]

Definition at line 124 of file jd_stop_eval.h.

4.67.4.27 double StopEval::SumYY [static, private]

Definition at line 125 of file jd_stop_eval.h.

4.67.4.28 DLList< double > StopEval::TotalStressList [static, private]

list of total stress values kept over the last lookback number of calls to stop eval. Min, Mean and slope are used over this list.

Definition at line 33 of file jd_stop_eval.C.

4.67.4.29 double StopEval::Variance [static, private]

Definition at line 126 of file jd_stop_eval.h.

4.67.4.30 bool StopEval::VerboseOn = false [static]

run-time ajdustable flag to provide verbose output.

Definition at line 71 of file globals.C.

4.67.4.31 jVector * StopEval::myPoints [private]

Definition at line 81 of file jd_stop_eval.h.

4.67.4.32 ofstream * StopEval::of [private]

Definition at line 80 of file jd_stop_eval.h.

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

- **jd_stop_eval.h**
- **globals.C**
- **jd_stop_eval.C**

4.68 UnitWeight Class Reference

`#include <graph_weights.h>`

Public Methods

- **UnitWeight** ()
- **UnitWeight** (double)
- **UnitWeight** (int)
- **UnitWeight** (bool)
- const UnitWeight& **operator=** (const UnitWeight &)
- **operator bool** () const
- **operator int** () const
- **operator double** () const
- int **Output** (ostream &os) const
- int **Input** (istream &)
- int **OutputBinary** (ostream &) const
- int **InputBinary** (istream &)
- **MAKECOMPARISONS_NO_COMPARE** (UnitWeight)
- **MAKE_IO_STREAM_OPERATORS** (UnitWeight)
- **MAKE_BIN_STREAM_OPERATORS** (UnitWeight)

4.68.1 Constructor & Destructor Documentation

4.68.1.1 UnitWeight::UnitWeight () [inline]

Definition at line 11 of file graph_weights.h.

```
00011                           {}
00012
```

4.68.1.2 UnitWeight::UnitWeight (double) [inline]

Definition at line 12 of file graph_weights.h.

```
00012                                 {}
00013
```

4.68.1.3 UnitWeight::UnitWeight (int) [inline]

Definition at line 13 of file graph_weights.h.

```
00013                                {}
00014
```

4.68.1.4 UnitWeight::UnitWeight (bool) [inline]

Definition at line 14 of file graph_weights.h.

```
00014                                 {}
00015
```

4.68.2 Member Function Documentation

4.68.2.1 int UnitWeight::Input (istream &) [inline]

Definition at line 26 of file graph_weights.h.

```
00027                   { return 1; }
00028
```

4.68.2.2 int UnitWeight::InputBinary (istream &) [inline]

Definition at line 30 of file graph_weights.h.

```
00031                   { return 1; }
00032
```

4.68.2.3 UnitWeight::MAKECOMPARISONS_NO_COMPARE (UnitWeight)

4.68.2.4 UnitWeight::MAKE_BIN_STREAM_OPERATORS (UnitWeight)

4.68.2.5 UnitWeight::MAKE_IO_STREAM_OPERATORS (UnitWeight)

4.68.2.6 int UnitWeight::Output (ostream & *os*) const [inline]

Definition at line 24 of file graph_weights.h.

```
00025                   { os << "_1_"; return 1; }
00026
```

4.68.2.7 int UnitWeight::OutputBinary (ostream &) const [inline]

Definition at line 28 of file graph_weights.h.

```
00029                   { return 1; }
00030
```

4.68.2.8 UnitWeight::operator bool () const [inline]

Definition at line 20 of file graph_weights.h.

```
00020                                   { return true; }
00021
```

4.68.2.9 UnitWeight::operator double () const [inline]

Definition at line 22 of file graph_weights.h.

```
00022                                     { return 1.0; }
00023
```

4.68.2.10 UnitWeight::operator int () const [inline]

Definition at line 21 of file graph_weights.h.

```
00021                                   { return 1; }
00022
```

4.68.2.11 const UnitWeight & UnitWeight::operator= (const UnitWeight &) [inline]

Definition at line 16 of file graph_weights.h.

```
00017                  { return *this; }
00018
```

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

- **graph_weights.h**

4.69 z Class Reference

`#include <randomgenerator.h>`

Public Methods

- **z** ()
- **~z** ()

Public Attributes

- int **seeded**
- double **u** [98]
- double **c**
- double **cd**
- double **cm**
- int **i97**
- int **j97**

4.69.1 Constructor & Destructor Documentation

4.69.1.1 z::z () [inline]

Definition at line 65 of file randomgenerator.h.

```
00065                   { seeded = 0; }
00066
```

4.69.1.2 z::~z () [inline]

Definition at line 66 of file randomgenerator.h.

```
00066                    {}
00067
```

4.69.2 Member Data Documentation

4.69.2.1 double z::c

Definition at line 69 of file randomgenerator.h.

4.69.2.2 double z::cd

Definition at line 69 of file randomgenerator.h.

4.69.2.3 double z::cm

Definition at line 69 of file randomgenerator.h.

4.69.2.4 int z::i97

Definition at line 70 of file randomgenerator.h.

4.69.2.5 int z::j97

Definition at line 70 of file randomgenerator.h.

4.69.2.6 int z::seeded

Definition at line 67 of file randomgenerator.h.

4.69.2.7 double z::u[98]

Definition at line 68 of file randomgenerator.h.

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

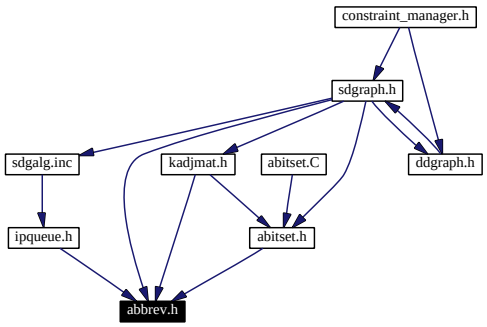
- **randomgenerator.h**

Chapter 5

GRAPE: Generalizable Replaceable Atomic Position Engine File Documentation

5.1 abbrev.h File Reference

This graph shows which files directly or indirectly include this file:



Typedefs

- typedef unsigned int **uint**
- typedef short int **sint**
- typedef long int **lint**
- typedef unsigned short int **usint**
- typedef unsigned long int **ulint**
- typedef unsigned char **uchar**
- typedef class **ABitSet ABS**

5.1.1 Typedef Documentation

5.1.1.1 typedef class ABitSet ABS

Definition at line 17 of file abbrev.h.

5.1.1.2 typedef long int lint

Definition at line 13 of file abbrev.h.

5.1.1.3 typedef short int sint

Definition at line 12 of file abbrev.h.

5.1.1.4 typedef unsigned char uchar

Definition at line 16 of file abbrev.h.

5.1.1.5 typedef unsigned int uint

Definition at line 11 of file abbrev.h.

5.1.1.6 typedef unsigned long int ulint

Definition at line 15 of file abbrev.h.

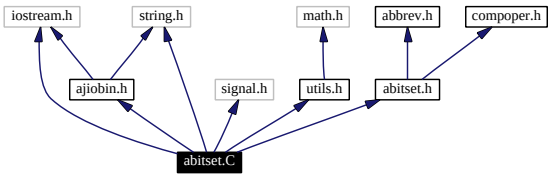
5.1.1.7 typedef unsigned short int usint

Definition at line 14 of file abbrev.h.

5.2 abitset.C File Reference

```
#include <iostream.h>
#include <string.h>
#include <signal.h>
#include "utils.h"
#include "ajiobin.h"
#include "abitset.h"
```

Include dependency graph for abitset.C:



Functions

- **uint b2c0** (**uint** i)
- **uint b2c1** (**uint** i)
- char **b2i** (**uint** i)
- ostream& **operator<<** (ostream &os, const **ABitSet** &abs)
- ostream& **OutputBinary** (ostream &os, const **ABitSet** &bs)
- istream& **InputBinary** (istream &is, **ABitSet** &bs)

Variables

- const char **all_set** [8] = {255,1,3,7,15,31,63,127}

5.2.1 Function Documentation

5.2.1.1 istream & InputBinary (istream & *is*, ABitSet & *bs*)

Definition at line 448 of file abitset.C.

Referenced by InputBinary(), and operator>=().

```
00449 {
00450         if (!is)
00451                 return is;
00452
00453         uint    n;
00454         InputBinary(is,n);
00455
00456         delete[] bs.data;
00457         bs.num = n;
00458         bs.data = new char[b2c1(n)];
00459         if (!bs.data)
00460                 raise(SIGQUIT);
00461
```

```
00462         is.read(bs.data, b2c1(bs.num));
00463         return is;
00464
```

5.2.1.2 ostream & OutputBinary (ostream & *os*, const ABitSet & *bs*)

Definition at line 436 of file abitset.C.

Referenced by OutputBinary(), and operator<=().

```
00437 {
00438         if (!os)
00439                 return os;
00440         OutputBinary(os,bs.num);
00441         if (!os)
00442                 return os;
00443         os.write(bs.data, b2c1(bs.num));
00444         return os;
00445
```

5.2.1.3 uint b2c0 (uint *i*) [inline]

Definition at line 13 of file abitset.C.

Referenced by ABitSet::Clear(), ABitSet::Invert(), ABitSet::Set(), and ABitSet::operator[]().

```
00014 {
00015         return i/8;
00016
```

5.2.1.4 uint b2c1 (uint *i*) [inline]

Definition at line 18 of file abitset.C.

Referenced by ABitSet::ABitSet(), ABitSet::ClearAll(), ABitSet::Compare(), ABitSet::Count(), ABitSet::Decrement(), ABitSet::Increment(), InputBinary(), ABitSet::InvertAll(), ABitSet::IsAllClear(), ABitSet::IsAllSet(), OutputBinary(), ABitSet::SetAll(), ABitSet::operator &=(), ABitSet::operator-=(), ABitSet::operator=(), ABitSet::operator^=(), and ABitSet::operator|=().

```
00019 {
00020         return (i+7)/8;
00021
```

5.2.1.5 char b2i (uint *i*) [inline]

Definition at line 23 of file abitset.C.

Referenced by ABitSet::Invert(), ABitSet::Set(), and ABitSet::operator[]().

```
00024 {
00025         return 1 << (i%8);
00026
```

5.2.1.6 ostream & operator<< (ostream & *os*, const ABitSet & *abs*)

Definition at line 367 of file abitset.C.

```
00368 {
00369         os << '<';
00370         for (uint i=0; i<abs.num; ++i)
00371                 os << (abs[i] ? '1' : '0');
00372         os << '>';
00373         return os;
00374
```

5.2.2 Variable Documentation

5.2.2.1 const char all_set[8] = {255,1,3,7,15,31,63,127} [static]

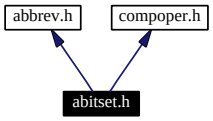
Definition at line 10 of file abitset.C.

5.3 abitset.h File Reference

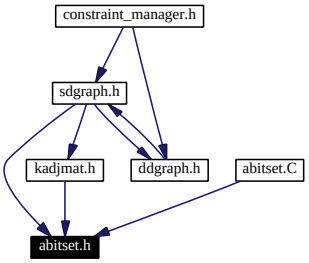
```
#include "abbrev.h"
```

```
#include "compoper.h"
```

Include dependency graph for abitset.h:



This graph shows which files directly or indirectly include this file:



Compounds

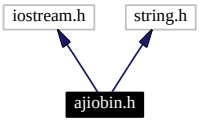
- class **ABitSet**
- class **ABitSet::Iterator**

5.4 ajiobin.h File Reference

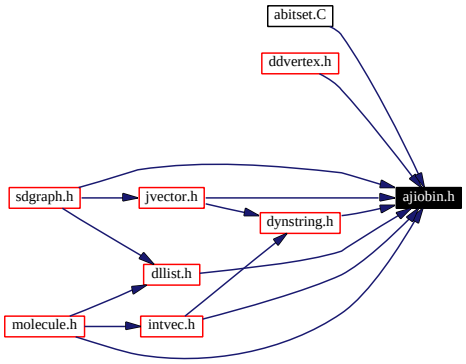
`#include <iostream.h>`

`#include <string.h>`

Include dependency graph for ajiobin.h:



This graph shows which files directly or indirectly include this file:



Functions

- ostream& **operator**<< (ostream &os, char c)
- ostream& **operator**<< (ostream &os, unsigned char c)
- ostream& **operator**<< (ostream &os, signed char c)
- ostream& **operator**<< (ostream &os, int i)
- ostream& **operator**<< (ostream &os, unsigned int i)
- ostream& **operator**<< (ostream &os, short s)
- ostream& **operator**<< (ostream &os, unsigned short s)
- ostream& **operator**<< (ostream &os, long l)
- ostream& **operator**<< (ostream &os, unsigned long l)
- ostream& **operator**<< (ostream &os, const char *cp)
- ostream& **operator**<< (ostream &os, bool b)
- ostream& **operator**<< (ostream &os, float f)
- ostream& **operator**<< (ostream &os, double d)
- ostream& **OutputBinary** (ostream &os, char c)
- ostream& **OutputBinary** (ostream &os, unsigned char c)
- ostream& **OutputBinary** (ostream &os, signed char c)
- ostream& **OutputBinary** (ostream &os, int i)
- ostream& **OutputBinary** (ostream &os, unsigned int i)
- ostream& **OutputBinary** (ostream &os, short s)
- ostream& **OutputBinary** (ostream &os, unsigned short s)

- ostream& **OutputBinary** (ostream &os, long l)
- ostream& **OutputBinary** (ostream &os, unsigned long l)
- ostream& **OutputBinary** (ostream &os, const char *cp)
- ostream& **OutputBinary** (ostream &os, bool b)
- ostream& **OutputBinary** (ostream &os, float f)
- ostream& **OutputBinary** (ostream &os, double d)
- istream& **operator>=** (istream &is, char &c)
- istream& **operator>=** (istream &is, unsigned char &c)
- istream& **operator>=** (istream &is, signed char &c)
- istream& **operator>=** (istream &is, int &i)
- istream& **operator>=** (istream &is, unsigned int &i)
- istream& **operator>=** (istream &is, short &s)
- istream& **operator>=** (istream &is, unsigned short &s)
- istream& **operator>=** (istream &is, long &l)
- istream& **operator>=** (istream &is, unsigned long &l)
- istream& **operator>=** (istream &is, char *cp)
- istream& **operator>=** (istream &is, bool &b)
- istream& **operator>=** (istream &is, float &f)
- istream& **operator>=** (istream &is, double &d)
- istream& **operator>=** (istream &is, long double &d)
- istream& **InputBinary** (istream &is, char &c)
- istream& **InputBinary** (istream &is, unsigned char &c)
- istream& **InputBinary** (istream &is, signed char &c)
- istream& **InputBinary** (istream &is, int &i)
- istream& **InputBinary** (istream &is, unsigned int &i)
- istream& **InputBinary** (istream &is, short &s)
- istream& **InputBinary** (istream &is, unsigned short &s)
- istream& **InputBinary** (istream &is, long &l)
- istream& **InputBinary** (istream &is, unsigned long &l)
- istream& **InputBinary** (istream &is, char *cp)
- istream& **InputBinary** (istream &is, bool &b)
- istream& **InputBinary** (istream &is, float &f)
- istream& **InputBinary** (istream &is, double &d)
- istream& **InputBinary** (istream &is, long double &d)

5.4.1 Function Documentation

5.4.1.1 istream & InputBinary (istream & *is*, long double & *d*) [inline]

Definition at line 352 of file ajiobin.h.

```
00353 {
00354         return is.read(&d,sizeof(long double));
00355
```

5.4.1.2 istream & InputBinary (istream & *is*, double & *d*) [inline]

Definition at line 347 of file ajiobin.h.

```
00348 {
00349         return is.read(&d,sizeof(double));
00350
```

5.4.1.3 istream & InputBinary (istream & *is*, float & *f*) [inline]

Definition at line 342 of file ajiobin.h.

```
00343 {
00344         return is.read(&f,sizeof(float));
00345
```

5.4.1.4 istream & InputBinary (istream & *is*, bool & *b*) [inline]

Definition at line 337 of file ajiobin.h.

```
00338 {
00339         return is.read(&b,sizeof(bool));
00340
```

5.4.1.5 istream & InputBinary (istream & *is*, char * *cp*) [inline]

Definition at line 329 of file ajiobin.h.

```
00330 {
00331         int     len;
00332         is.read(&len,sizeof(int));
00333         is.read(cp,len);
00334         return is;
00335
```

5.4.1.6 istream & InputBinary (istream & *is*, unsigned long & *l*) [inline]

Definition at line 324 of file ajiobin.h.

```
00325 {
00326         return is.read(&l,sizeof(unsigned long));
00327
```

5.4.1.7 istream & InputBinary (istream & *is*, long & *l*) [inline]

Definition at line 319 of file ajiobin.h.

```
00320 {
00321         return is.read(&l,sizeof(long));
00322
```

5.4.1.8 istream & InputBinary (istream & *is*, unsigned short & *s*) [inline]

Definition at line 314 of file ajiobin.h.

```
00315 {
00316         return is.read(&s,sizeof(unsigned short));
00317
```

5.4.1.9 istream & InputBinary (istream & *is*, short & *s*) [inline]

Definition at line 309 of file ajiobin.h.

```
00310 {
00311         return is.read(&s,sizeof(short));
00312
```

5.4.1.10 istream & InputBinary (istream & *is*, unsigned int & *i*) [inline]

Definition at line 304 of file ajiobin.h.

```
00305 {
00306         return is.read(&i,sizeof(unsigned int));
00307
```

5.4.1.11 istream & InputBinary (istream & *is*, int & *i*) [inline]

Definition at line 299 of file ajiobin.h.

```
00300 {
00301         return is.read(&i,sizeof(int));
00302
```

5.4.1.12 istream & InputBinary (istream & *is*, signed char & *c*) [inline]

Definition at line 294 of file ajiobin.h.

```
00295 {
00296         return is.read(&c,sizeof(signed char));
00297
```

5.4.1.13 istream & InputBinary (istream & *is*, unsigned char & *c*) [inline]

Definition at line 289 of file ajiobin.h.

```
00290 {
00291         return is.read(&c,sizeof(unsigned char));
00292
```

5.4.1.14 istream & InputBinary (istream & *is*, char & *c*) [inline]

Definition at line 284 of file ajiobin.h.

```
00285 {
00286         return is.read(&c,sizeof(char));
00287
```

5.4.1.15 ostream & OutputBinary (ostream & *os*, double *d*) [inline]

Definition at line 188 of file ajiobin.h.

```
00189 {
00190         os.write(&d,sizeof(double));
00191         return os;
00192 
```

5.4.1.16 ostream & OutputBinary (ostream & *os*, float *f*) [inline]

Definition at line 182 of file ajiobin.h.

```
00183 {
00184         os.write(&f,sizeof(float));
00185         return os;
00186 
```

5.4.1.17 ostream & OutputBinary (ostream & *os*, bool *b*) [inline]

Definition at line 176 of file ajiobin.h.

```
00177 {
00178         os.write(&b,sizeof(bool));
00179         return os;
00180 
```

5.4.1.18 ostream & OutputBinary (ostream & *os*, const char * *cp*) [inline]

Definition at line 168 of file ajiobin.h.

```
00169 {
00170         int     len = strlen(cp)+1;
00171         os.write(&len,sizeof(int));
00172         os.write(cp,len);
00173         return os;
00174 
```

5.4.1.19 ostream & OutputBinary (ostream & *os*, unsigned long *l*) [inline]

Definition at line 162 of file ajiobin.h.

```
00163 {
00164         os.write(&l,sizeof(unsigned long));
00165         return os;
00166 
```

5.4.1.20 ostream & OutputBinary (ostream & *os*, long *l*) [inline]

Definition at line 156 of file ajiobin.h.

```
00157 {
00158         os.write(&l,sizeof(long));
00159         return os;
00160 
```

5.4.1.21 ostream & OutputBinary (ostream & *os*, unsigned short *s*) [inline]

Definition at line 150 of file ajiobin.h.

```
00151 {
00152         os.write(&s,sizeof(unsigned short));
00153         return os;
00154 
```

5.4.1.22 ostream & OutputBinary (ostream & *os*, short *s*) [inline]

Definition at line 144 of file ajiobin.h.

```
00145 {
00146         os.write(&s,sizeof(short));
00147         return os;
00148 
```

5.4.1.23 ostream & OutputBinary (ostream & *os*, unsigned int *i*) [inline]

Definition at line 138 of file ajiobin.h.

```
00139 {
00140         os.write(&i,sizeof(unsigned int));
00141         return os;
00142 
```

5.4.1.24 ostream & OutputBinary (ostream & *os*, int *i*) [inline]

Definition at line 132 of file ajiobin.h.

```
00133 {
00134         os.write(&i,sizeof(int));
00135         return os;
00136 
```

5.4.1.25 ostream & OutputBinary (ostream & *os*, signed char *c*) [inline]

Definition at line 126 of file ajiobin.h.

```
00127 {
00128         os.write(&c,sizeof(signed char));
00129         return os;
00130 
```

5.4.1.26 ostream & OutputBinary (ostream & *os*, unsigned char *c*) [inline]

Definition at line 120 of file ajiobin.h.

```
00121 {
00122         os.write(&c,sizeof(unsigned char));
00123         return os;
00124 
```

5.4.1.27 ostream & OutputBinary (ostream & *os*, char *c*) [inline]

Definition at line 114 of file ajiobin.h.

```
00115 {
00116         os.write(&c,sizeof(char));
00117         return os;
00118 
```

5.4.1.28 ostream & operator<= (ostream & *os*, double *d*) [inline]

Definition at line 98 of file ajiobin.h.

```
00099 {
00100         os.write(&d,sizeof(double));
00101         return os;
00102 
```

5.4.1.29 ostream & operator<= (ostream & *os*, float *f*) [inline]

Definition at line 92 of file ajiobin.h.

```
00093 {
00094         os.write(&f,sizeof(float));
00095         return os;
00096 
```

5.4.1.30 ostream & operator<= (ostream & *os*, bool *b*) [inline]

Definition at line 86 of file ajiobin.h.

```
00087 {
00088         os.write(&b,sizeof(bool));
00089         return os;
00090 
```

5.4.1.31 ostream & operator<= (ostream & *os*, const char * *cp*) [inline]

Definition at line 78 of file ajiobin.h.

```
00079 {
00080         int     len = strlen(cp)+1;
00081         os.write(&len,sizeof(int));
00082         os.write(cp,len);
00083         return os;
00084 
```

5.4.1.32 ostream & operator<= (ostream & *os*, unsigned long *l*) [inline]

Definition at line 72 of file ajiobin.h.

```
00073 {
00074         os.write(&l,sizeof(unsigned long));
00075         return os;
00076 
```

5.4.1.33 ostream & operator<= (ostream & *os*, long *l*) [inline]

Definition at line 66 of file ajiobin.h.

```
00067 {
00068         os.write(&l,sizeof(long));
00069         return os;
00070 
```

5.4.1.34 ostream & operator<= (ostream & *os*, unsigned short *s*) [inline]

Definition at line 60 of file ajiobin.h.

```
00061 {
00062         os.write(&s,sizeof(unsigned short));
00063         return os;
00064 
```

5.4.1.35 ostream & operator<= (ostream & *os*, short *s*) [inline]

Definition at line 54 of file ajiobin.h.

```
00055 {
00056         os.write(&s,sizeof(short));
00057         return os;
00058 
```

5.4.1.36 ostream & operator<= (ostream & *os*, unsigned int *i*) [inline]

Definition at line 48 of file ajiobin.h.

```
00049 {
00050         os.write(&i,sizeof(unsigned int));
00051         return os;
00052 
```

5.4.1.37 ostream & operator<= (ostream & *os*, int *i*) [inline]

Definition at line 42 of file ajiobin.h.

```
00043 {
00044         os.write(&i,sizeof(int));
00045         return os;
00046 
```

5.4.1.38 ostream & operator<= (ostream & *os*, signed char *c*) [inline]

Definition at line 36 of file ajiobin.h.

```
00037 {
00038         os.write(&c,sizeof(signed char));
00039         return os;
00040 
```

5.4.1.39 ostream & operator<= (ostream & *os*, unsigned char *c*) [inline]

Definition at line 30 of file ajiobin.h.

```
00031 {
00032         os.write(&c,sizeof(unsigned char));
00033         return os;
00034
```

5.4.1.40 ostream & operator<= (ostream & *os*, char *c*) [inline]

Definition at line 24 of file ajiobin.h.

```
00025 {
00026         os.write(&c,sizeof(char));
00027         return os;
00028
```

5.4.1.41 istream & operator>= (istream & *is*, long double & *d*) [inline]

Definition at line 275 of file ajiobin.h.

```
00276 {
00277         return is.read(&d,sizeof(long double));
00278
```

5.4.1.42 istream & operator>= (istream & *is*, double & *d*) [inline]

Definition at line 270 of file ajiobin.h.

```
00271 {
00272         return is.read(&d,sizeof(double));
00273
```

5.4.1.43 istream & operator>= (istream & *is*, float & *f*) [inline]

Definition at line 265 of file ajiobin.h.

```
00266 {
00267         return is.read(&f,sizeof(float));
00268
```

5.4.1.44 istream & operator>= (istream & *is*, bool & *b*) [inline]

Definition at line 260 of file ajiobin.h.

```
00261 {
00262         return is.read(&b,sizeof(bool));
00263
```

5.4.1.45 istream & operator>= (istream & *is*, char * *cp*) [inline]

Definition at line 252 of file ajiobin.h.

```
00253 {
00254         int     len;
00255         is.read(&len,sizeof(int));
00256         is.read(cp,len);
00257         return is;
00258
```

5.4.1.46 istream & operator>= (istream & *is*, unsigned long & *l*) [inline]

Definition at line 247 of file ajiobin.h.

```
00248 {
00249         return is.read(&l,sizeof(unsigned long));
00250
```

5.4.1.47 istream & operator>= (istream & *is*, long & *l*) [inline]

Definition at line 242 of file ajiobin.h.

```
00243 {
00244         return is.read(&l,sizeof(long));
00245
```

5.4.1.48 istream & operator>= (istream & *is*, unsigned short & *s*) [inline]

Definition at line 237 of file ajiobin.h.

```
00238 {
00239         return is.read(&s,sizeof(unsigned short));
00240
```

5.4.1.49 istream & operator>= (istream & *is*, short & *s*) [inline]

Definition at line 232 of file ajiobin.h.

```
00233 {
00234         return is.read(&s,sizeof(short));
00235
```

5.4.1.50 istream & operator>= (istream & *is*, unsigned int & *i*) [inline]

Definition at line 227 of file ajiobin.h.

```
00228 {
00229         return is.read(&i,sizeof(unsigned int));
00230
```

5.4.1.51 istream & operator>= (istream & *is*, int & *i*) [inline]

Definition at line 222 of file ajiobin.h.

```
00223 {
00224         return is.read(&i,sizeof(int));
00225
```

5.4.1.52 istream & operator>= (istream & *is*, signed char & *c*) [inline]

Definition at line 217 of file ajiobin.h.

```
00218 {
00219         return is.read(&c,sizeof(signed char));
00220
```

5.4.1.53 istream & operator>= (istream & *is*, unsigned char & *c*) [inline]

Definition at line 212 of file ajiobin.h.

```
00213 {
00214         return is.read(&c,sizeof(unsigned char));
00215
```

5.4.1.54 istream & operator>= (istream & *is*, char & *c*) [inline]

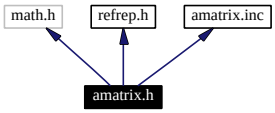
Definition at line 207 of file ajiobin.h.

```
00208 {
00209         return is.read(&c,sizeof(char));
00210
```

5.5 amatrix.h File Reference

```
#include <math.h>
#include <refrep.h>
#include "amatrix.inc"
```

Include dependency graph for amatrix.h:

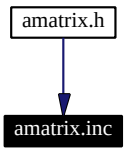


Compounds

- class **AMatrix**
- class **AMatrix::AMatrixRep**

5.6 amatrix.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- ostream& **operator**<< (ostream &os, const **AMatrix** &m)
- istream& **operator**>> (istream &is, **AMatrix** &m)

5.6.1 Function Documentation

5.6.1.1 ostream & operator<< (ostream & *os*, const AMatrix & *m*) [inline]

Definition at line 404 of file amatrix.inc.

```
00405 { return OutputBinary(os,m); }
00406
```

5.6.1.2 istream & operator>> (istream & *is*, AMatrix & *m*) [inline]

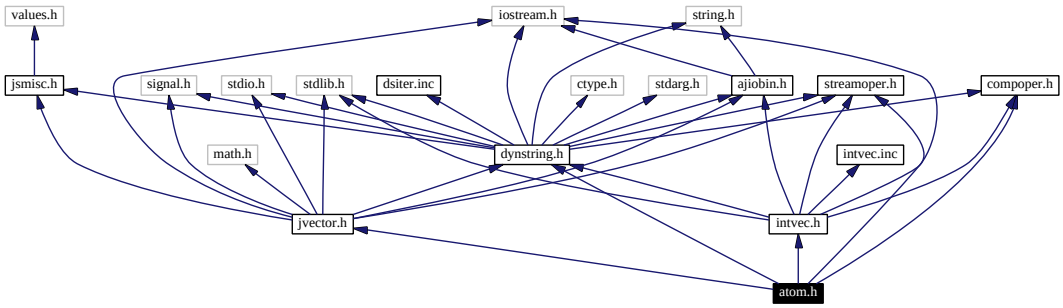
Definition at line 408 of file amatrix.inc.

```
00409 { return InputBinary(is,m); }
00410
```

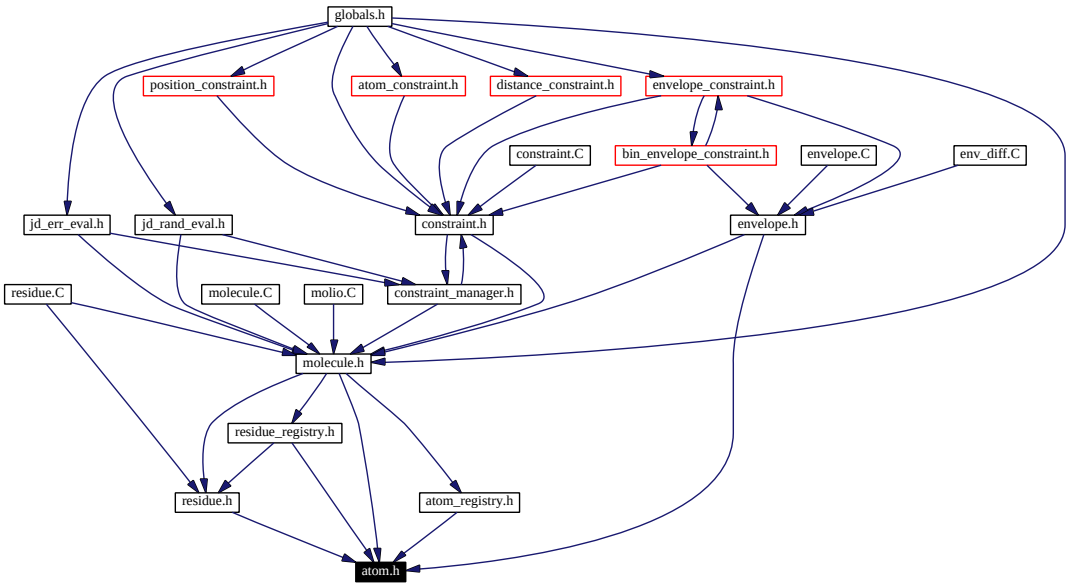
5.7 atom.h File Reference

```
#include "dynstring.h"
#include "intvec.h"
#include "jvector.h"
#include "compoper.h"
#include "streamoper.h"
```

Include dependency graph for atom.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Atom**

Defines

- #define **UNINITIALIZED_ATOM_ID** -1

- #define **MAX_NUM_BONDS** 14;

5.7.1 Define Documentation

5.7.1.1 #define MAX_NUM_BONDS 14;

Definition at line 29 of file atom.h.

5.7.1.2 #define UNINITIALIZED_ATOM_ID -1

//

File: **atom.h**

Used to save information about one atom.

Positions are no longer stored in **Atom** (p. 43)' instead the molecule is resposible for them.

So now **Atom** (p. 43) class doesn't hold any useful information except ID Eventually It will hold VDW radius, atom type, local perturbation settings, chain identifier, residue/nucleotide number – all things intrinsic to the atom. (-jd)

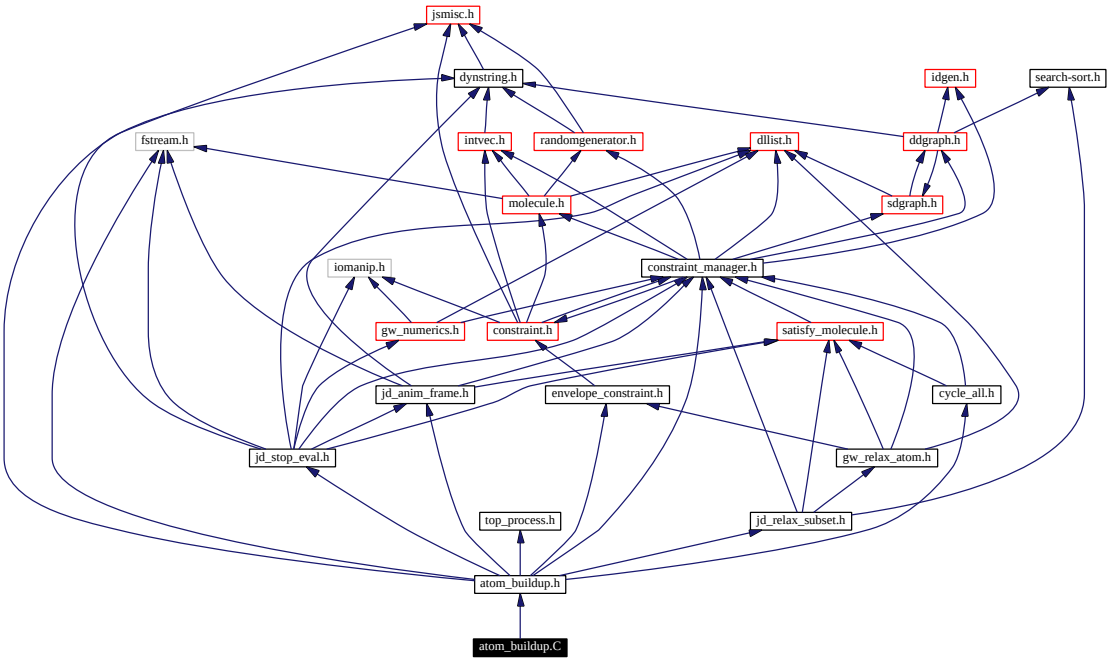
//

Definition at line 28 of file atom.h.

5.8 atom_buildup.C File Reference

```
#include "atom_buildup.h"
```

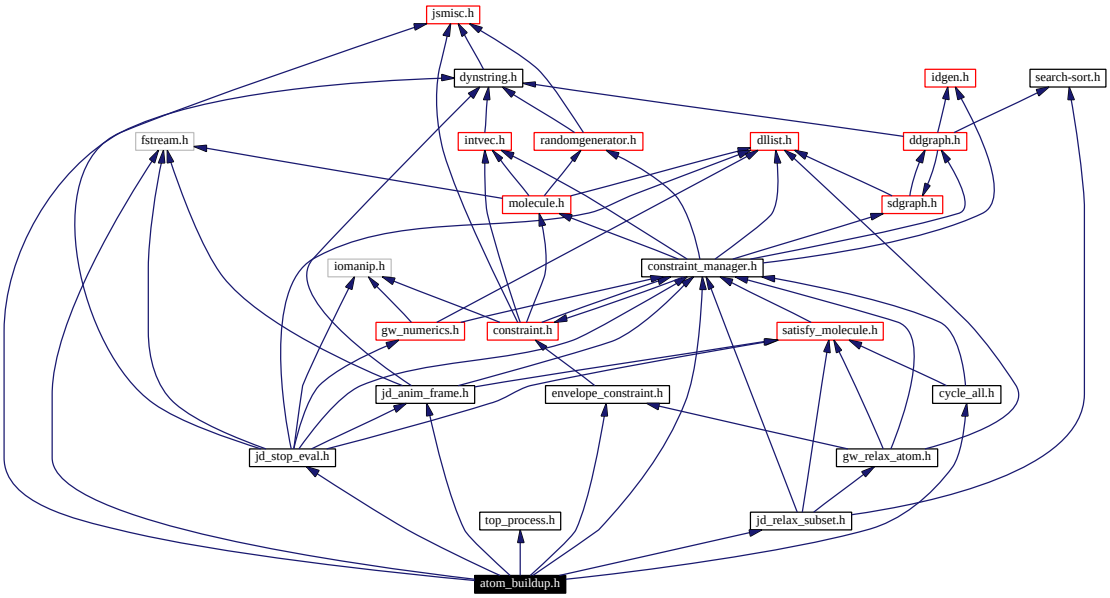
Include dependency graph for atom_buildup.C:



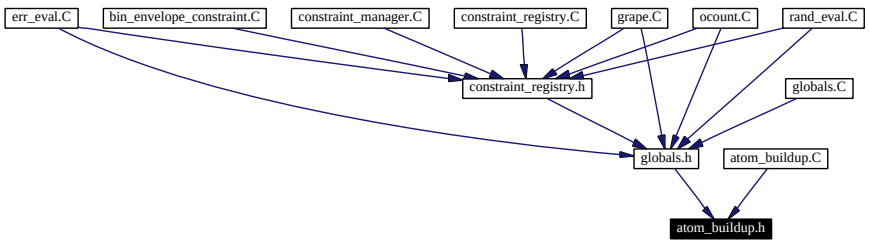
5.9 atom_buildup.h File Reference

```
#include <fstream.h>
#include "constraint_manager.h"
#include "top_process.h"
#include "dynstring.h"
#include "cycle_all.h"
#include "jd_stop_eval.h"
#include "jd_anim_frame.h"
#include "jd_relax_subset.h"
#include "envelope_constraint.h"
```

Include dependency graph for atom_buildup.h:



This graph shows which files directly or indirectly include this file:



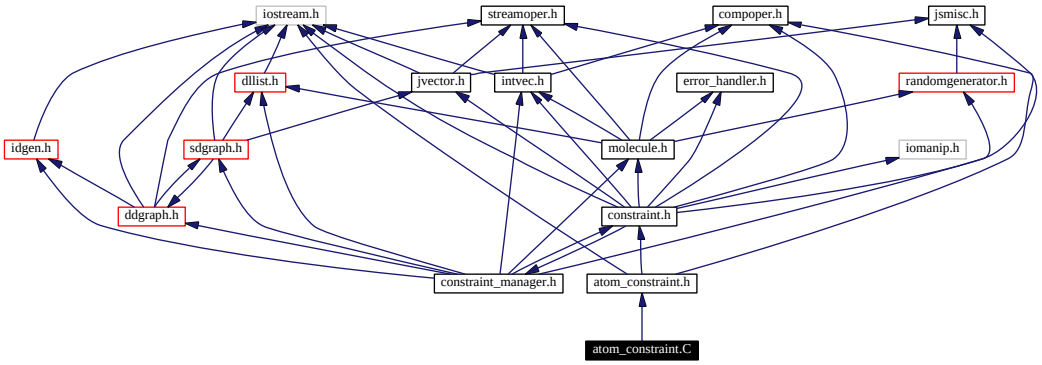
Compounds

- class **AtomBuildup**

5.10 atom_constraint.C File Reference

```
#include "atom_constraint.h"
```

Include dependency graph for atom_constraint.C:



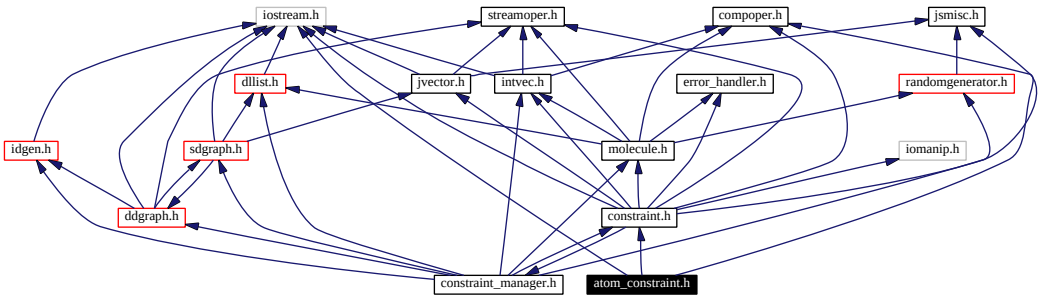
5.11 atom_constraint.h File Reference

```
#include <iostream.h>
```

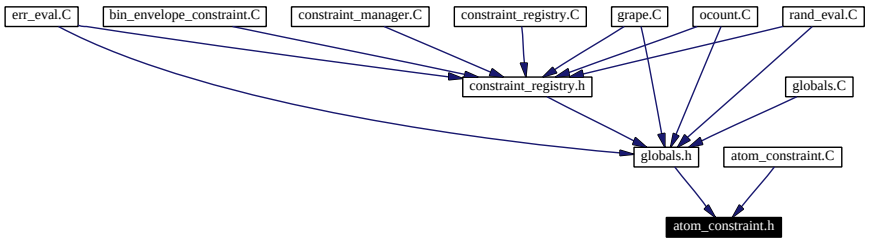
```
#include "constraint.h"
```

```
#include "compoper.h"
```

Include dependency graph for atom_constraint.h:



This graph shows which files directly or indirectly include this file:



Compounds

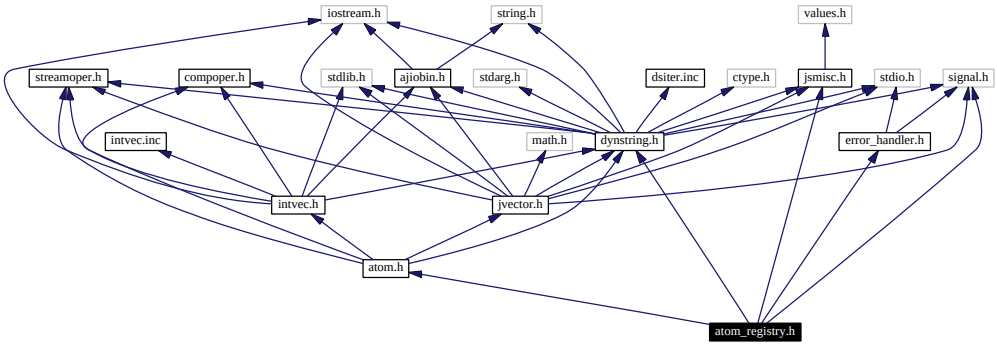
- class **AtomConstraint**

5.12 atom_registry.C File Reference

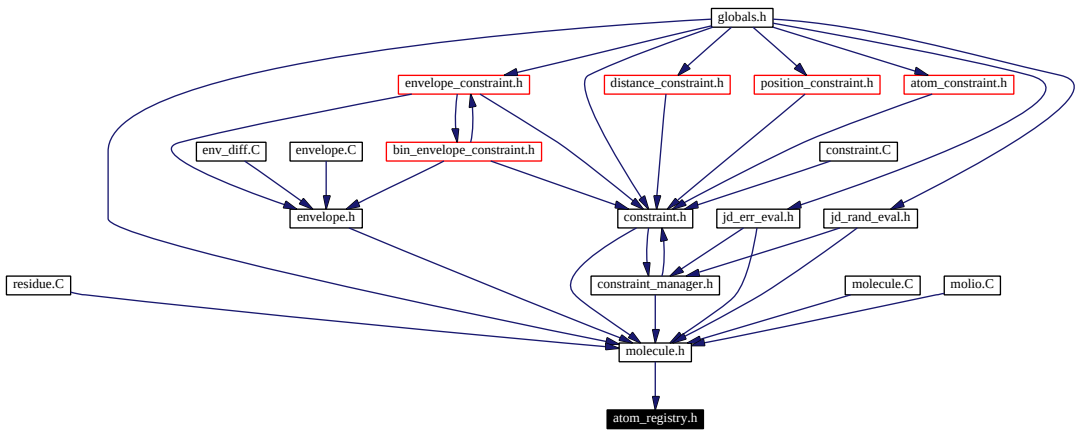
5.13 atom_registry.h File Reference

```
#include <signal.h>
#include "dynstring.h"
#include "atom.h"
#include "jsmisc.h"
#include "error_handler.h"
```

Include dependency graph for atom_registry.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **AtomRegistry**

Defines

- #define **num_known_atom_species** 10

5.13.1 Define Documentation

5.13.1.1 #define num_known_atom_species 10

//

File: **atom_registry.h**

//

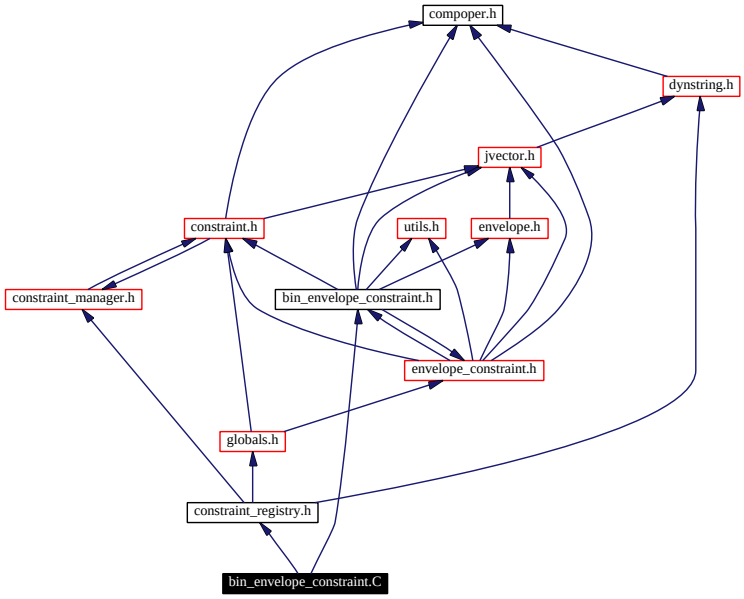
Definition at line 20 of file atom_registry.h.

5.14 bin_envelope_constraint.C File Reference

```
#include "bin_envelope_constraint.h"
```

```
#include "constraint_registry.h"
```

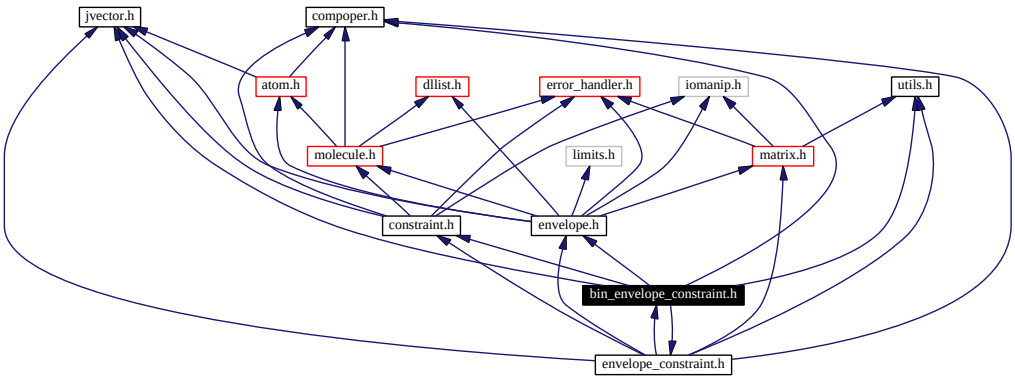
Include dependency graph for bin_envelope_constraint.C:



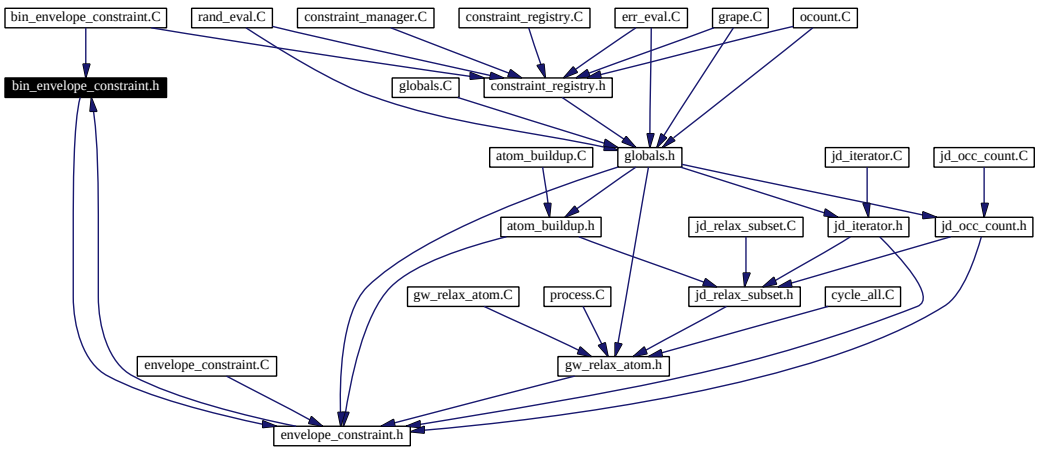
5.15 bin_envelope_constraint.h File Reference

```
#include "envelope.h"
#include "envelope_constraint.h"
#include "constraint.h"
#include "compoper.h"
#include "jvector.h"
#include "utils.h"
```

Include dependency graph for bin_envelope_constraint.h:



This graph shows which files directly or indirectly include this file:



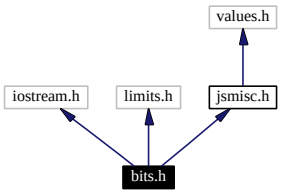
Compounds

- class **BinEnvelopeConstraint**

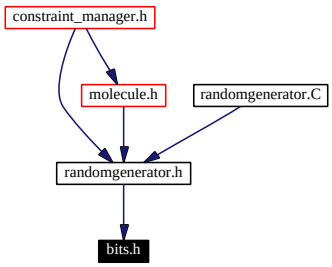
5.16 bits.h File Reference

```
#include <iostream.h>
#include <limits.h>
#include "jsmisc.h"
```

Include dependency graph for bits.h:



This graph shows which files directly or indirectly include this file:



Functions

- **uint reverse_bits** (**uint** i)
- ostream& **print_bits** (ostream &os, **uint** v)

5.16.1 Function Documentation

5.16.1.1 ostream & print_bits (ostream & *os*, uint *v*) [inline]

Definition at line 25 of file bits.h.

```
00026 {
00027   uint mask=(uint)1<<31;
00028   for(uint i=1; i<=32; ++i)
00029     {
00030       os << ((v & mask) == 0) ? 0 : 1;
00031       v <<=1;
00032       if(i%8 == 0 && i != 32)
00033           os << ' ';
00034     }
00035
```

5.16.1.2 uint reverse_bits (uint *i*) [inline]

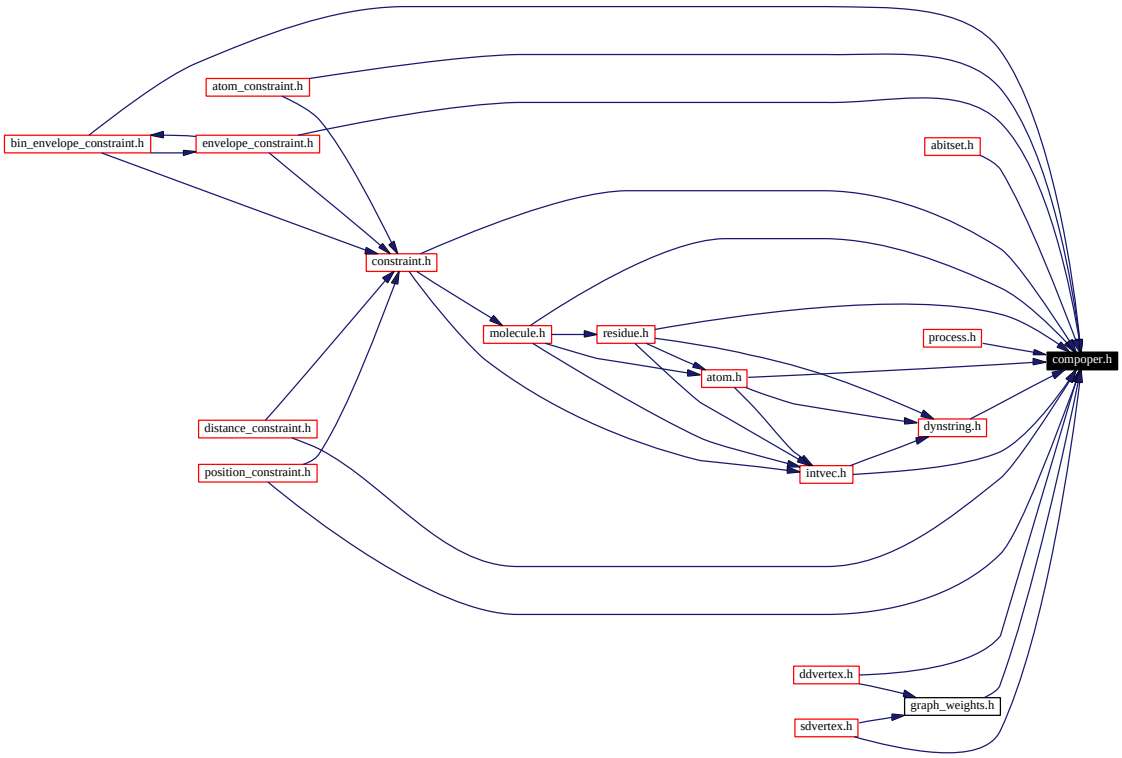
Definition at line 9 of file bits.h.

Referenced by RanBaseClass::Seed().

```
00010   {
00011     uint rev=0;
00012     uint numbits = (uint) sizeof(uint)*8;
00013     int bit;
00014
00015     for(uint j=0; j<numbits; ++j)
00016       {
00017         bit = i & 0x00000001;
00018         bit <<= numbits-j-1;
00019         rev |= bit;
00020         i >>= 1;
00021       }
00022
```

5.17 compoper.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define **MAKECOMPARISONS_NO_COMPARE**(CLASSTYPE_XX)
- #define **MAKECOMPARISONS_WITH_COMPARE**(CLASSTYPE_XX)

5.17.1 Define Documentation

5.17.1.1 #define MAKECOMPARISONS_NO_COMPARE(CLASSTYPE_XX)

Value:

```
bool    operator == (const CLASSTYPE_XX& co) const \
        { return this == &co; } \
bool    operator != (const CLASSTYPE_XX& co) const \
        { return this != &co; } \
bool    operator <= (const CLASSTYPE_XX& co) const \
        { return this <= &co; } \
bool    operator >= (const CLASSTYPE_XX& co) const \
        { return this >= &co; } \
bool    operator < (const CLASSTYPE_XX& co) const \
        { return this < &co; } \
bool    operator > (const CLASSTYPE_XX& co) const \
        { return this > &co; }
```

Definition at line 14 of file compoper.h.

5.17.1.2 #define MAKECOMPARISONS_WITH_COMPARE(CLASSTYPE_XX)

Value:

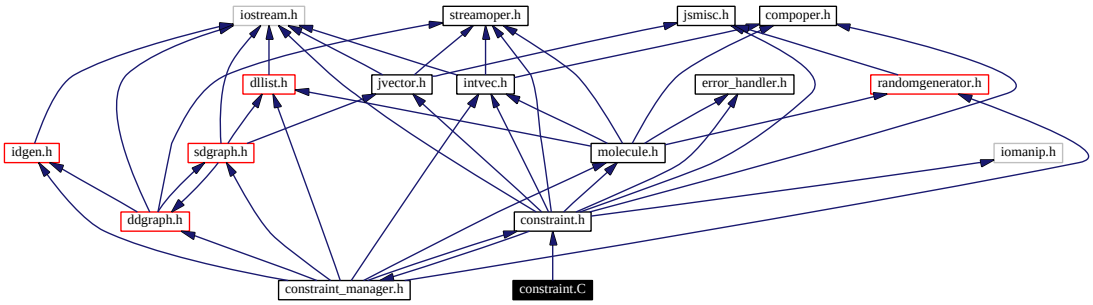
```
bool    operator == (const CLASSTYPE_XX& co) const \
        { return Compare(co) == 0; } \
bool    operator != (const CLASSTYPE_XX& co) const \
        { return Compare(co) != 0; } \
bool    operator <= (const CLASSTYPE_XX& co) const \
        { return Compare(co) <= 0; } \
bool    operator >= (const CLASSTYPE_XX& co) const \
        { return Compare(co) >= 0; } \
bool    operator < (const CLASSTYPE_XX& co) const \
        { return Compare(co) < 0; } \
bool    operator > (const CLASSTYPE_XX& co) const \
        { return Compare(co) > 0; }
```

Definition at line 28 of file compoper.h.

5.18 constraint.C File Reference

```
#include "constraint.h"
```

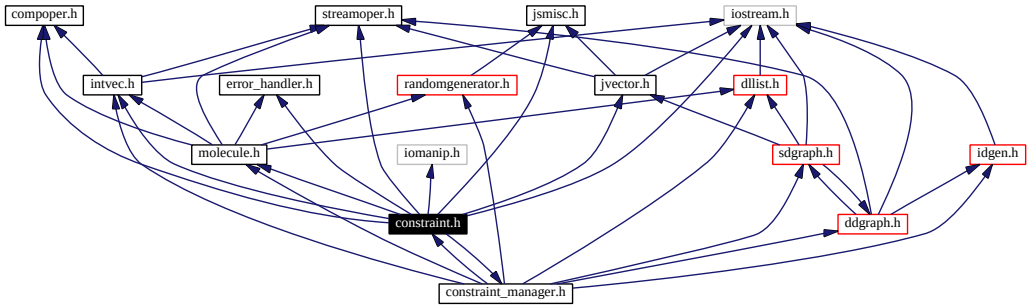
Include dependency graph for constraint.C:



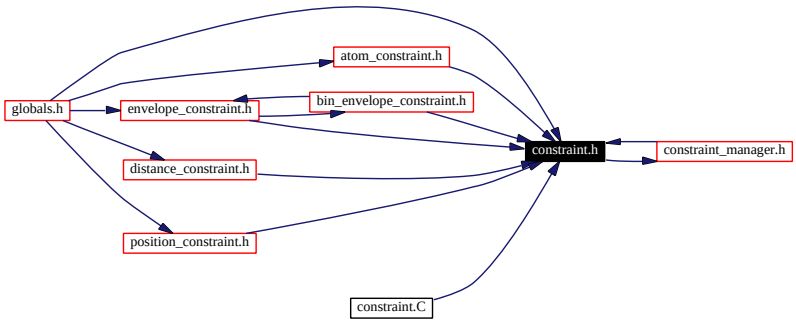
5.19 constraint.h File Reference

```
#include <iostream.h>
#include <iomanip.h>
#include "constraint_manager.h"
#include "molecule.h"
#include "jvector.h"
#include "intvec.h"
#include "compoper.h"
#include "streamoper.h"
#include "jsmisc.h"
#include "error_handler.h"
```

Include dependency graph for constraint.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Constraint**

Defines

- #define **UNINITIALIZED_CONSTRAINT_ID** -1

- #define __CONSTRAINT_DEFINED__

5.19.1 Define Documentation

5.19.1.1 #define UNINITIALIZED_CONSTRAINT_ID -1

//

File: **constraint.h**

Abstract constraint class. All other (real) constraint types are classes that inherit from this class.

//

Definition at line 27 of file constraint.h.

5.19.1.2 #define __CONSTRAINT_DEFINED__

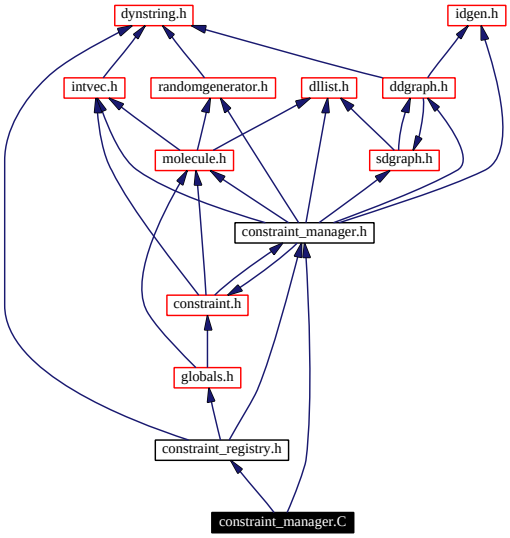
Value:

Definition at line 325 of file constraint.h.

5.20 constraint_manager.C File Reference

```
#include "constraint_manager.h"
#include "constraint_registry.h"
```

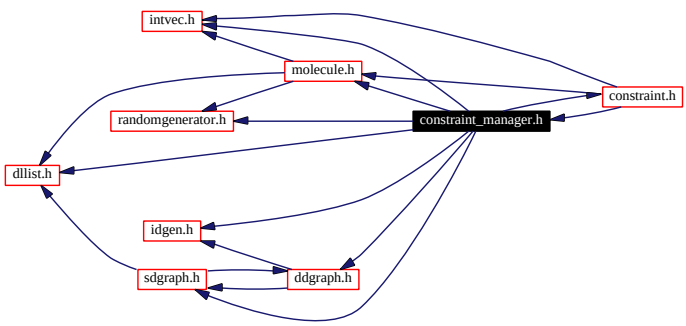
Include dependency graph for constraint_manager.C:



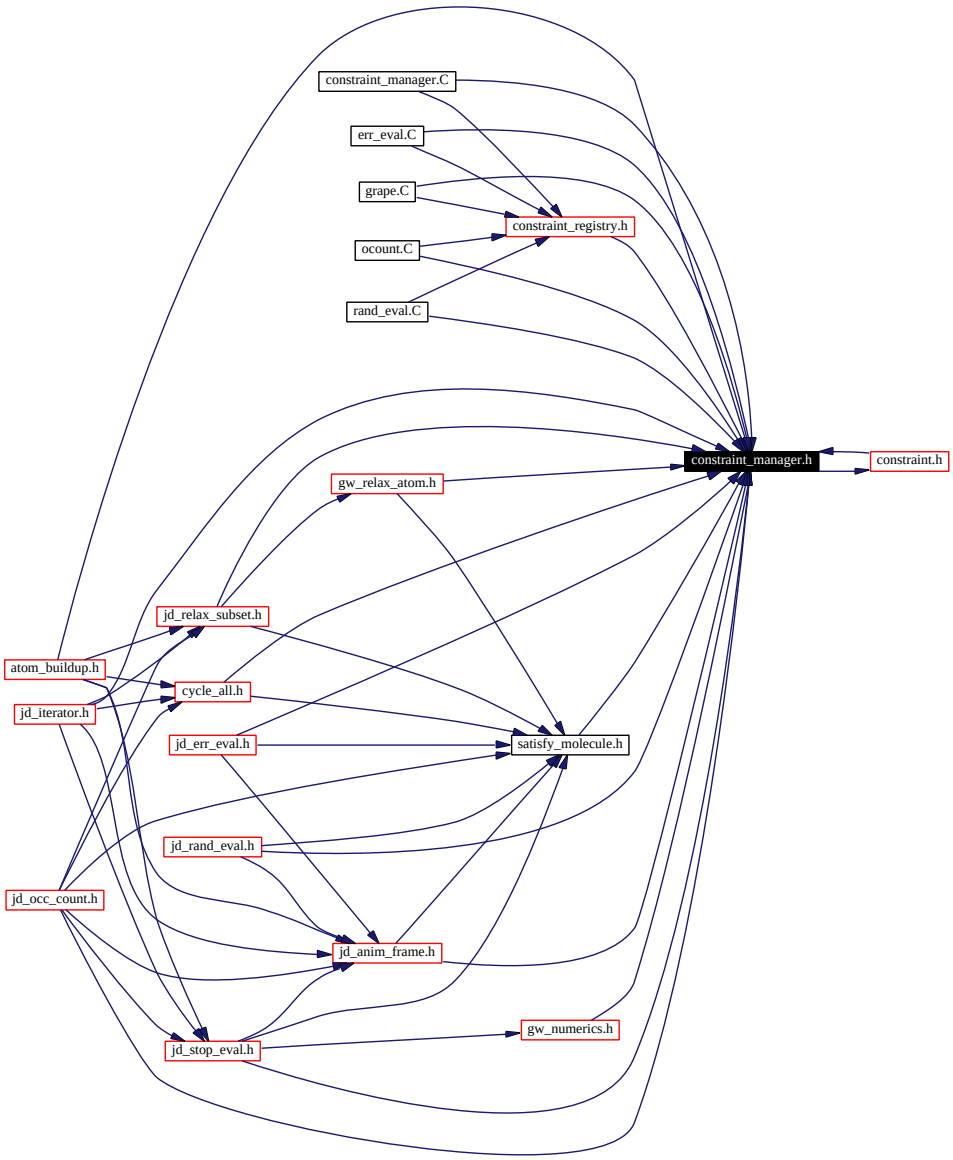
5.21 constraint_manager.h File Reference

```
#include "dllist.h"
#include "intvec.h"
#include "molecule.h"
#include "ddgraph.h"
#include "sdgraph.h"
#include "idgen.h"
#include "randomgenerator.h"
#include <constraint.h>
```

Include dependency graph for constraint_manager.h:



This graph shows which files directly or indirectly include this file:



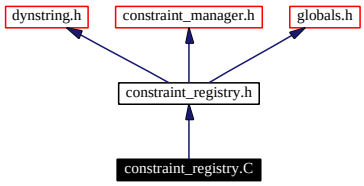
Compounds

- class **ConstraintManager**

5.22 constraint_registry.C File Reference

```
#include "constraint_registry.h"
```

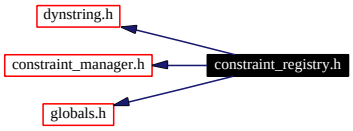
Include dependency graph for constraint_registry.C:



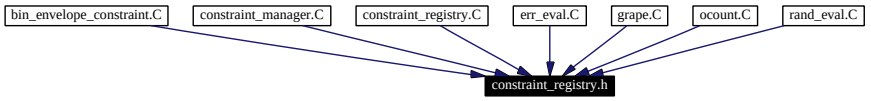
5.23 constraint_registry.h File Reference

```
#include "dynstring.h"
#include "constraint_manager.h"
#include "globals.h"
```

Include dependency graph for constraint_registry.h:



This graph shows which files directly or indirectly include this file:



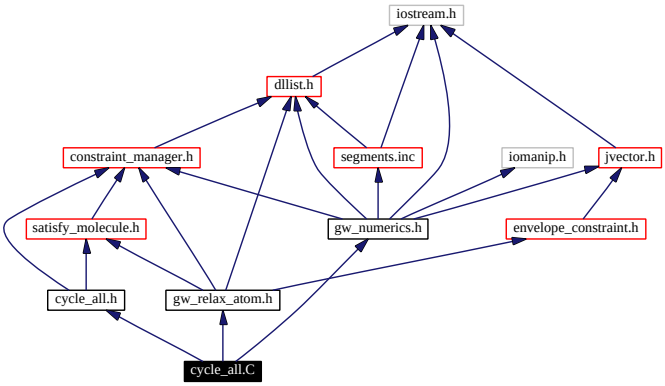
Compounds

- class **ConstraintRegistry**

5.24 cycle_all.C File Reference

```
#include "cycle_all.h"
#include "gw_numerics.h"
#include "gw_relax_atom.h"
```

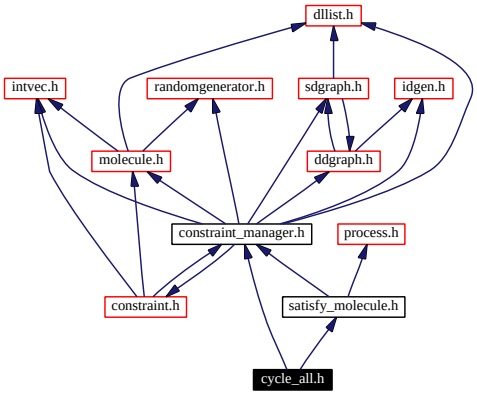
Include dependency graph for cycle_all.C:



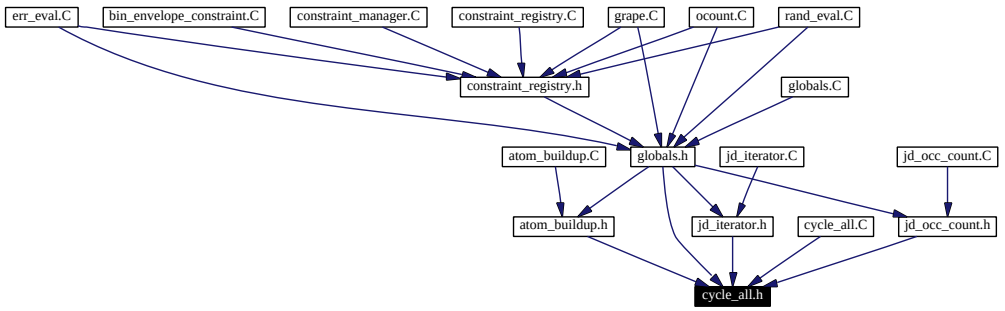
5.25 cycle_all.h File Reference

```
#include "constraint_manager.h"
#include "satisfy_molecule.h"
```

Include dependency graph for cycle_all.h:



This graph shows which files directly or indirectly include this file:



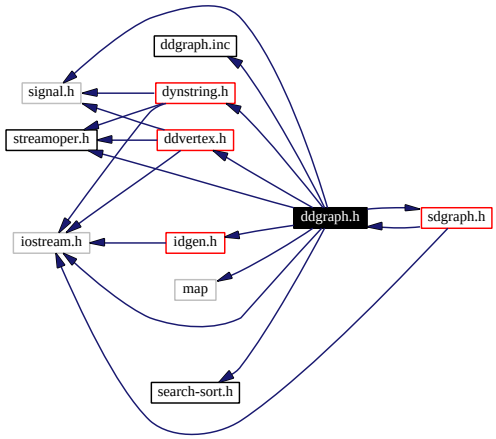
Compounds

- class **CycleAll**

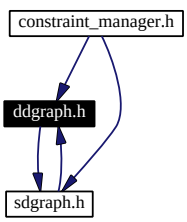
5.26 ddgraph.h File Reference

```
#include <iostream.h>
#include <map>
#include <signal.h>
#include "idgen.h"
#include "dynstring.h"
#include "search-sort.h"
#include "streamoper.h"
#include "ddvertex.h"
#include "sdgraph.h"
#include <ddgraph.inc>
```

Include dependency graph for ddgraph.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ddGraph**

Enumerations

- enum **GraphStatus** { **NoVertices** = -10 , **VertexBad** = - 9 , **VertexInG** = - 8 , **VertexNotInG** = - 7 , **VertexAddFailure** = - 6 , **NoEdges** = - 5 , **EdgeBad** = - 4 , **EdgeInG** = - 3 , **EdgeNotInG** = - 2 , **EdgeAddFailure** = - 1 }

Functions

- ostream& **PrintGStatus** (ostream &os, const **GraphStatus** &gs)

Variables

- const **DynString V_BEGIN_DELIM** = "{"
- const **DynString V_END_DELIM** = "}"
- const **DynString E_BEGIN_DELIM** = "["
- const **DynString E_END_DELIM** = "]"

5.26.1 Enumeration Type Documentation

5.26.1.1 enum GraphStatus

Enumeration values:

NoVertices

VertexBad

VertexInG

VertexNotInG

VertexAddFailure

NoEdges

EdgeBad

EdgeInG

EdgeNotInG

EdgeAddFailure

Definition at line 47 of file ddgraph.h.

```
00048 {
00049         NoVertices              = -10 ,
00050         VertexBad               = - 9 ,
00051         VertexInG               = - 8 ,
00052         VertexNotInG            = - 7 ,
00053         VertexAddFailure        = - 6 ,
00054         NoEdges                 = - 5 ,
00055         EdgeBad                 = - 4 ,
00056         EdgeInG                 = - 3 ,
00057         EdgeNotInG              = - 2 ,
00058         EdgeAddFailure          = - 1
00059
```

5.26.2 Function Documentation

5.26.2.1 ostream & PrintGStatus (ostream & *os*, const GraphStatus & *gs*) [inline]

Definition at line 1 of file ddgraph.inc.

```
00002   {
00003     os << "<<";
00004     if(gs >= 0) os << "Operation successful.";
00005     else
00006       {
00007         switch(gs)
00008           {
00009             case NoVertices:
00010                 os << "There were no vertices in G!";
00011                 break;
00012             case VertexBad:
00013                 os << "NULL vertex access attempted!";
00014                 break;
00015             case VertexInG:
00016                 os << "The vertex was already in G!";
00017                 break;
00018             case VertexNotInG:
00019                 os << "The vertex was not found in G!";
00020                 break;
00021             case VertexAddFailure:
00022                 os << "The vertex could not be added to G!";
00023                 break;
00024             case NoEdges:
00025                 os << "There were no edges in G!";
00026                 break;
00027             case EdgeBad:
00028                 os << "NULL edge access attempted!";
00029                 break;
00030             case EdgeInG:
00031                 os << "The edge was already in G!";
00032                 break;
00033             case EdgeNotInG:
00034                 os << "The edge was not found in G!";
00035                 break;
00036             case EdgeAddFailure:
00037                 os << "The edge could not be added to G!";
00038                 break;
00039             default:
00040                 os << "Some unkown error occurred!";
00041                 break;
00042           }
00043       }
00044
```

5.26.3 Variable Documentation

5.26.3.1 const DynString E_BEGIN_DELIM = "["

Definition at line 63 of file ddgraph.h.

5.26.3.2 const DynString E_END_DELIM = "]"

Definition at line 64 of file ddgraph.h.

5.26.3.3 const DynString V_BEGIN_DELIM = "{"

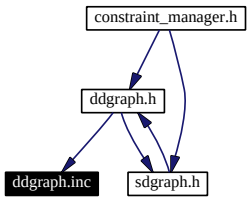
Definition at line 61 of file ddgraph.h.

5.26.3.4 const DynString V_END_DELIM = "}"

Definition at line 62 of file ddgraph.h.

5.27 ddgraph.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- ostream& **PrintGStatus** (ostream &os, const **GraphStatus** &gs)
- istream& **InputBinary** (istream &is, **ddGraph**< DT, WT > &ddg)
- ostream& **OutputBinary** (ostream &os, const **ddGraph**< DT, WT > &ddg)
- istream& **operator**>= (istream &is, **ddGraph**< DT, WT > &ddg)
- template<class DT, class WT> ostream& **operator**<= (ostream &os, const **ddGraph**< DT, WT > &ddg)

5.27.1 Function Documentation

5.27.1.1 istream & InputBinary (istream & *is*, ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 632 of file ddgraph.inc.

```
00633   {
00634     sdGraph<DT,WT> G;
00635     if(InputBinary(is,G).bad())
00636         return is;
00637     ddg = G;
00638     return is;
00639
```

5.27.1.2 ostream & OutputBinary (ostream & *os*, const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 642 of file ddgraph.inc.

```
00643   {
00644     sdGraph<DT,WT> G(ddg);
00645     return OutputBinary(os,G);
00646
```

5.27.1.3 ostream& PrintGStatus (ostream & *os*, const GraphStatus & *gs*) [inline]

Definition at line 1 of file ddgraph.inc.

```
00002   {
00003     os << "<<";
00004     if(gs >= 0) os << "Operation successful.";
00005     else
00006       {
00007         switch(gs)
00008           {
00009             case NoVertices:
00010                 os << "There were no vertices in G!";
00011                 break;
00012             case VertexBad:
00013                 os << "NULL vertex access attempted!";
00014                 break;
00015             case VertexInG:
00016                 os << "The vertex was already in G!";
00017                 break;
00018             case VertexNotInG:
00019                 os << "The vertex was not found in G!";
00020                 break;
00021             case VertexAddFailure:
00022                 os << "The vertex could not be added to G!";
00023                 break;
00024             case NoEdges:
00025                 os << "There were no edges in G!";
00026                 break;
00027             case EdgeBad:
00028                 os << "NULL edge access attempted!";
00029                 break;
00030             case EdgeInG:
00031                 os << "The edge was already in G!";
00032                 break;
00033             case EdgeNotInG:
00034                 os << "The edge was not found in G!";
00035                 break;
00036             case EdgeAddFailure:
00037                 os << "The edge could not be added to G!";
00038                 break;
00039             default:
00040                 os << "Some unkown error occurred!";
00041                 break;
00042           }
00043       }
00044
```

5.27.1.4 template<class DT, class WT> ostream & operator<= (ostream & *os*, const ddGraph< DT, WT > & *ddg*) [inline]

Definition at line 653 of file ddgraph.inc.

```
00654   { return OutputBinary(os,ddg); }
00655
```

5.27.1.5 istream & operator>= (istream & *is*, ddGraph< DT, WT > & *ddg*) [inline]

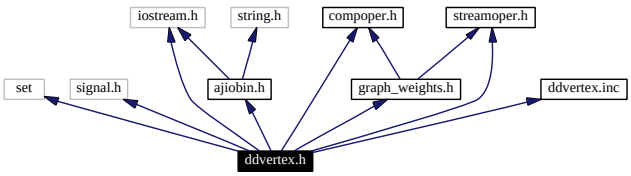
Definition at line 649 of file ddgraph.inc.

```
00650   { return IntputBinary(is,ddg); }
00651
```

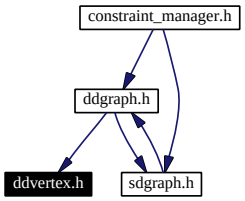
5.28 ddvertex.h File Reference

```
#include <set>
#include <signal.h>
#include <iostream.h>
#include "ajiobin.h"
#include "compoper.h"
#include "streamoper.h"
#include "graph_weights.h"
#include <ddvertex.inc>
```

Include dependency graph for ddvertex.h:



This graph shows which files directly or indirectly include this file:

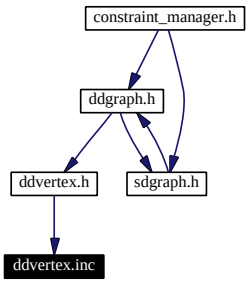


Compounds

- class **ddEdge**
- class **ddVertex**

5.29 ddvertex.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class WT> ostream& **operator**<< (ostream &os, const **ddVertex**< WT > &vert)

5.29.1 Function Documentation

5.29.1.1 template<class WT> ostream & operator<< (ostream & *os*, const ddVertex< WT > & *vert*) [inline]

Definition at line 121 of file ddvertex.inc.

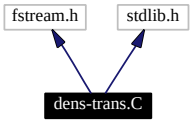
```
00122   {
00123     os << '\t' << vert.ID << '\n'
00124        << '\t' << '>' << ' ' << vert.OutE.size() << '\n';
00125     vert.PrintOutEdges(os);
00126     os << '\t' << '<' << ' ' << vert.InE.size()  << '\n';
00127     vert.PrintInEdges(os);
00128     return os;
00129
```

5.30 dens-trans.C File Reference

`#include <fstream.h>`

`#include <stdlib.h>`

Include dependency graph for dens-trans.C:



Functions

- int **main** (int argc, char **argv)

5.30.1 Function Documentation

5.30.1.1 int main (int *argc*, char ** *argv*)

Definition at line 4 of file dens-trans.C.

```
00004                                {
00005
00006   if (argc != 2) {
00007     cerr << "must supply the filename as arg 1" << endl;
00008     exit (1);
00009   }
00010
00011   ifstream in(argv[1]);
00012
00013   if (!in){
00014     cerr << "Unable to open " << argv[1] << endl;
00015     exit (2);
00016   }
00017
00018   char d[255];
00019   char temp[255];
00020
00021   int xdim, ydim, zdim;
00022   double minx, miny, minz;
00023   double maxx, maxy, maxz;
00024
00025   in.getline(d,255); // # AVS
00026   cout << d << endl;
00027   in.getline(d,255); // ndim
00028   cout << d << endl;
00029   //dim1
00030   in.ignore(5);
00031   in >>  xdim;
00032   cout << "xdim= "<< xdim << endl;
00033   //dim2
00034   in.ignore(6); // +\n
00035   in >>  ydim;
00036   cout << "ydim= "<< ydim << endl;
00037   //dim3
00038   in.ignore(6); // +\n
00039   in >>  zdim;
```

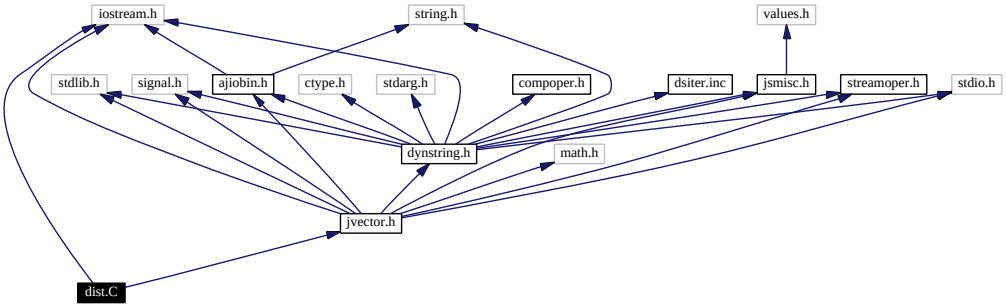
```
00040   cout << "zdim= "<< zdim << endl;
00041
00042   cerr << "dim: " << xdim << ", "<< ydim << ", "<< zdim << endl;
00043
00044   in.getline(d,255); // \n
00045
00046   in.getline(d,255); // nspace
00047   cout << d << endl;
00048   in.getline(d,255); // vclean
00049   cout << d << endl;
00050   in.getline(d,255); // data
00051   cout << d << endl;
00052   in.getline(d,255); // field
00053   cout << d << endl;
00054
00055   // mins
00056   in.ignore (9);
00057   in >>  minx >> miny >> minz;
00058   cout << "min_ext= " << minx << " " << miny << " " << minz << endl;
00059   cerr << "min: " <<  minx << ", " << miny << ", " << minz << endl;
00060
00061   // maxs
00062   in.ignore (10);
00063   in >>  maxx >> maxy >> maxz;
00064   cout << "max_ext= " << maxx << " " << maxy << " " << maxz << endl;
00065   cerr << "max: "  <<  maxx << ", " << maxy << ", " << maxz << endl;
00066
00067   in.getline(d,255); // nspace
00068
00069   in.getline(d,255); // # end of header
00070   cout << d << endl;
00071   in.ignore(2); // ^L^L  #!!##! WHATEVER -- drop these;
00072
00073   float val;
00074   long read = 0;
00075
00076   for (int zh =0;zh<zdim ;zh++){
00077     for (int yh =0;yh<ydim ;yh++){
00078       for (int xh =0;xh<xdim ;xh++){
00079         in.read(&val,sizeof(float));
00080         if (!in.good()){
00081           cerr << "reading error!\n read "<< read << " entries, expected "<< xdim*ydim*zdim << endl;
00082           exit(3);
00083         }
00084         read++;
00085         cout << val << " ";
00086       }
00087     }
00088   }
00089
```

5.31 dist.C File Reference

```
#include <iostream.h>
```

```
#include "jvector.h"
```

Include dependency graph for dist.C:



Functions

- int **main** (int argc, char **argv)

5.31.1 Function Documentation

5.31.1.1 int main (int *argc*, char ** *argv*)

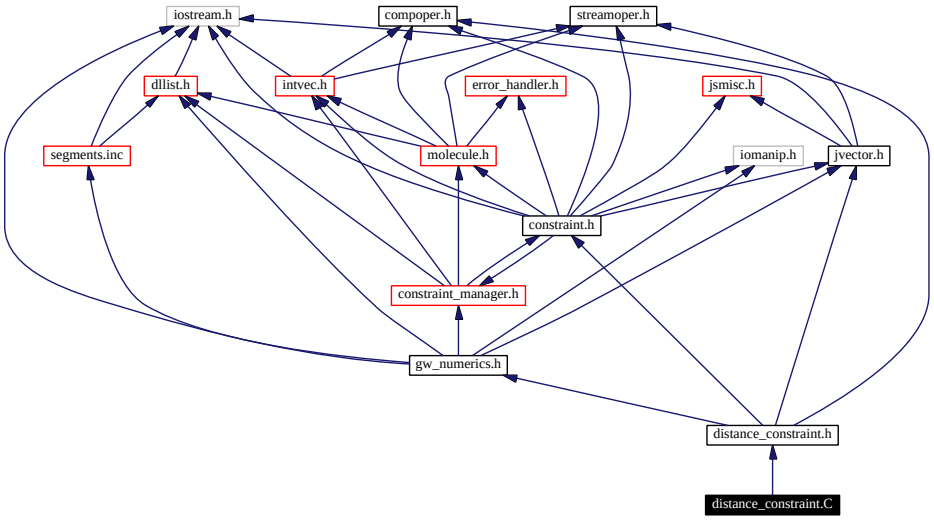
Definition at line 5 of file dist.C.

```
00006   {
00007     jVector v1;
00008     jVector v2;
00009
00010     while (1){
00011
00012       cout << "Vector1:";
00013       cin >> v1;
00014       cout << "Vector2:";
00015       cin >> v2;
00016       cout << "dist= " << euclid(v1,v2) << endl << endl;
00017
00018
00019     }
00020
```

5.32 distance_constraint.C File Reference

```
#include "distance_constraint.h"
```

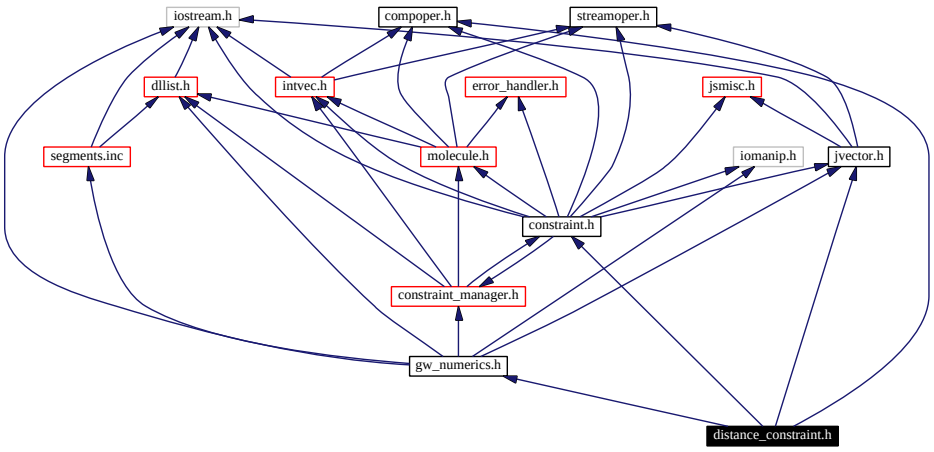
Include dependency graph for distance_constraint.C:



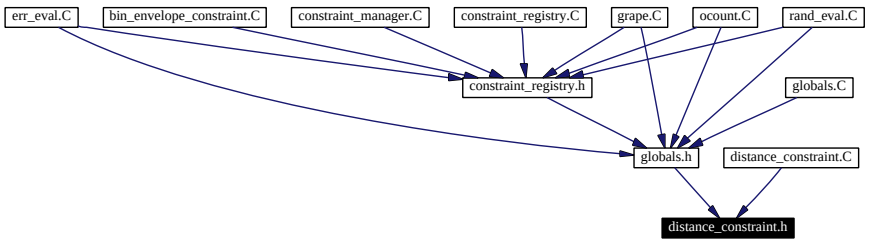
5.33 distance_constraint.h File Reference

```
#include "constraint.h"
#include "compoper.h"
#include "jvector.h"
#include "gw_numerics.h"
```

Include dependency graph for distance_constraint.h:



This graph shows which files directly or indirectly include this file:



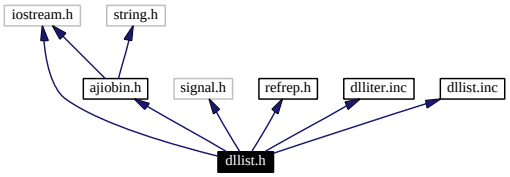
Compounds

- class **DistanceConstraint**

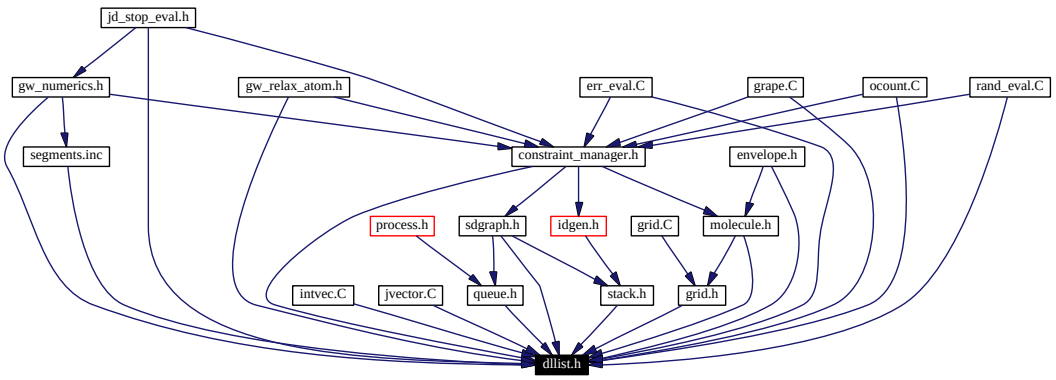
5.34 dllist.h File Reference

```
#include <iostream.h>
#include <signal.h>
#include "refrep.h"
#include "ajiobin.h"
#include <dlliter.inc>
#include "dllist.inc"
```

Include dependency graph for dllist.h:



This graph shows which files directly or indirectly include this file:

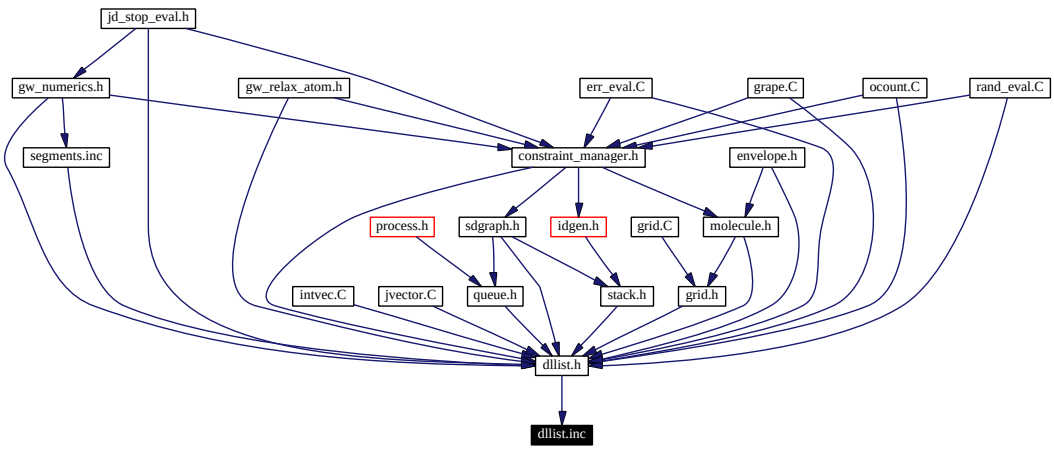


Compounds

- class **DLList**
- class **DLListNode**
- class **DLListRep**

5.35 dllist.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class T> ostream& **operator**<< (ostream &os, const **DLList**< T > &dll)
- istream& **InputBinary** (istream &is, **DLList**< T > &dll)
- ostream& **OutputBinary** (ostream &os, const **DLList**< T > &dll)
- istream& **operator**>= (istream &is, **DLList**< T > &dll)
- template<class T> ostream& **operator**<= (ostream &os, const **DLList**< T > &dll)

5.35.1 Function Documentation

5.35.1.1 istream & InputBinary (istream & *is*, DLList< T > & *dll*) [inline]

Definition at line 389 of file dllist.inc.

```
00390   {
00391     // Delete everything:
00392     //
00393     dll.Clear();
00394 
00395     // Read in the number of elements:
00396     //
00397     int numel;
00398     if(InputBinary(is,numel).bad())
00399         return is;
00400 
00401     T stuff;
00402     for(int i=0; i<numel; ++i)
00403       {
00404         if(InputBinary(is,stuff).bad())
00405             return is;
00406         dll.AddToBack(stuff);
00407       }
00408 
```

5.35.1.2 ostream & OutputBinary (ostream & *os*, const DLList< T > & *dll*) [inline]

Definition at line 412 of file dllist.inc.

```
00413   {
00414     // Write the number of elements:
00415     //
00416     if(OutputBinary(os, dll.Number()).bad())
00417         return os;
00418
00419     for(DLList<T>::Iterator it = dll.GetIterator(); it; ++it)
00420         if(OutputBinary(os, *it).bad())
00421             return os;
00422     return os;
00423
```

5.35.1.3 template<class T> ostream & operator<< (ostream & *os*, const DLList< T > & *dll*) [inline]

Definition at line 385 of file dllist.inc.

```
00386   { return dll.Print(os); }
00387
```

5.35.1.4 template<class T> ostream & operator<= (ostream & *os*, const DLList< T > & *dll*) [inline]

Definition at line 431 of file dllist.inc.

```
00432   { return OutputBinary(os,dll); }
00433
```

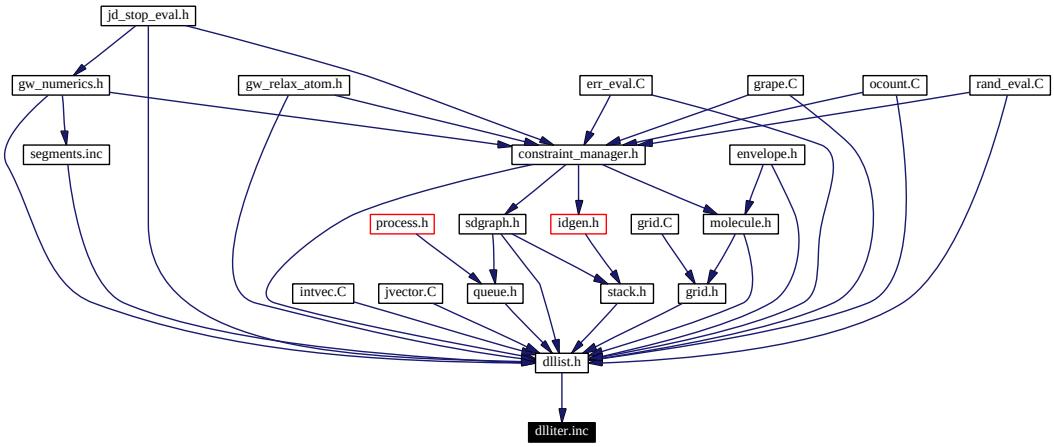
5.35.1.5 istream & operator>= (istream & *is*, DLList< T > & *dll*) [inline]

Definition at line 427 of file dllist.inc.

```
00428   { return InputBinary(os,dll); }
00429
```

5.36 dlliter.inc File Reference

This graph shows which files directly or indirectly include this file:



Compounds

- class **ConstIterator**
- class **Iterator**

5.37 dsiter.inc File Reference

This graph shows which files directly or indirectly include this file:

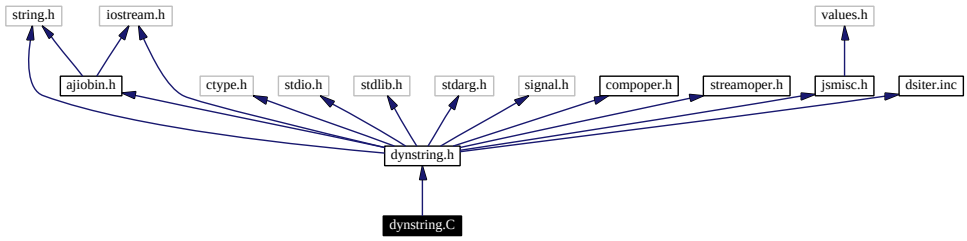
Compounds

- class **DSIterator**

5.38 dynstring.C File Reference

```
#include "dynstring.h"
```

Include dependency graph for dynstring.C:



Functions

- void **SetDSDelim** (const **DynString** &delim)
- void **AddDSDelim** (char delim)
- void **RemDSDelim** (char delim)
- **DynString GetDSDelim** ()
- **DynString IntToDS** (int i)
- **DynString Reproduce** (char c, int n)

5.38.1 Function Documentation

5.38.1.1 void AddDSDelim (char *delim*)

Definition at line 45 of file dynstring.C.

```
00046   { DynString::Delimiter[(uint) delim] = 1; }
00047
```

5.38.1.2 DynString GetDSDelim ()

Definition at line 51 of file dynstring.C.

Referenced by DSIterator::DSIterator(), jVector::dynread(), and IntVec::dynread().

```
00052   {
00053     DynString delim;
00054     delim.DestResizeToFit(256);
00055     int j=0;
00056     for(uint i=0; i<256; ++i)
00057       {
00058         if(DynString::Delimiter[i])
00059           {
00060             delim.innards[j] = (char) i;
00061             ++j;
00062           }
00063       }
00064
```

5.38.1.3 DynString IntToDS (int *i*)

Definition at line 351 of file dynstring.C.
Referenced by jdIterator::run(), and AtomBuildup::run().

```
00352   {
00353     char s[100]; s[0] = '\0';
00354     sprintf(s, "%i", i);
00355     DynString ds(s);
00356     return ds;
00357
```

5.38.1.4 void RemDSDelim (char *delim*)

Definition at line 47 of file dynstring.C.

```
00048   { DynString::Delimiter[(uint) delim] = 0; }
00049
```

5.38.1.5 DynString Reproduce (char *c*, int *n*)

Definition at line 745 of file dynstring.C.

```
00746   {
00747     DynString str;
00748     if(!str.DestResizeToFit(n)) return str;
00749     for(int i=0; i<n; ++i) str.innards[i] = c;
00750     str.innards[n] = '\0';
00751     str.L = n;
00752     return str;
00753
```

5.38.1.6 void SetDSDelim (const DynString & *delim*)

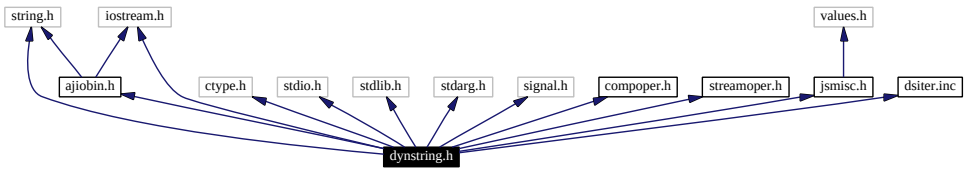
Definition at line 34 of file dynstring.C.
Referenced by jVector::dynread(), and IntVec::dynread().

```
00035   {
00036     uint j;
00037     for(int i=0; i<256; ++i) DynString::Delimiter[i] = 0;
00038     for(int i=0; i<delim.L; ++i)
00039       {
00040         j = (uint) delim.innards[i];
00041         DynString::Delimiter[j] = 1;
00042       }
00043
```

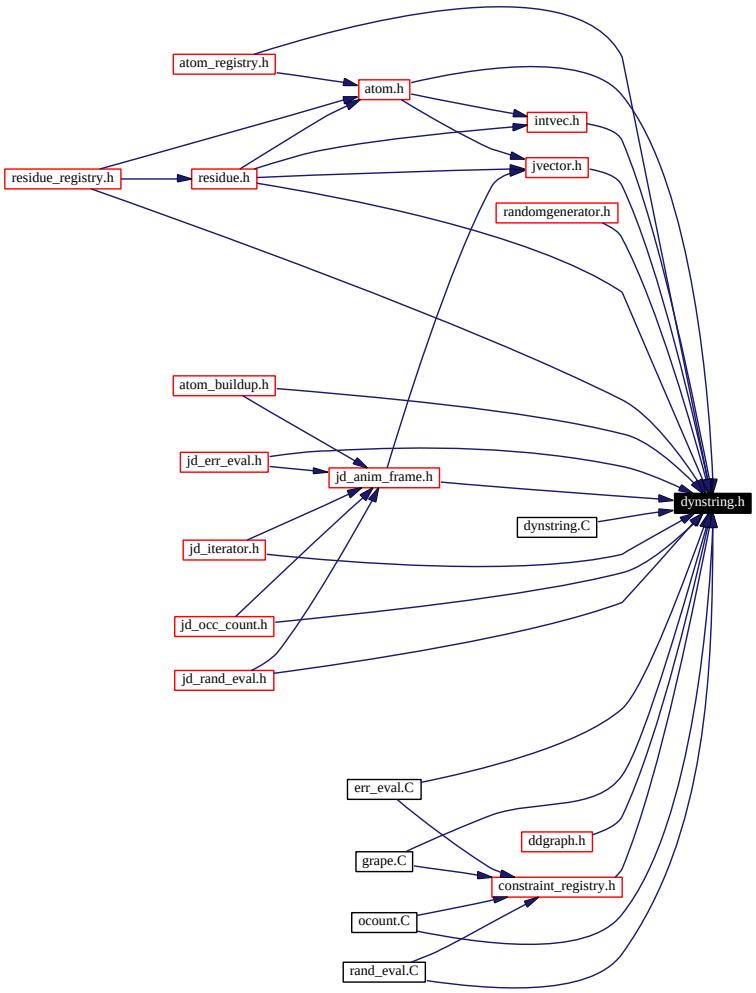
5.39 dynstring.h File Reference

```
#include <string.h>
#include <ctype.h>
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <signal.h>
#include <iostream.h>
#include "compoper.h"
#include "streamoper.h"
#include "jsmisc.h"
#include "ajiobin.h"
#include <dsiter.inc>
```

Include dependency graph for dynstring.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **DynString**

Typedefs

- typedef **DynString DS**

5.39.1 Typedef Documentation

5.39.1.1 typedef DynString DS

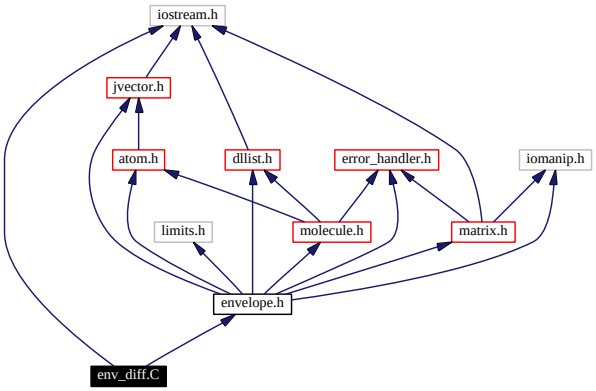
Definition at line 247 of file dynstring.h.

5.40 env_diff.C File Reference

```
#include <iostream.h>
```

```
#include "envelope.h"
```

Include dependency graph for env_diff.C:



Functions

- int **main** (int argc, char **argv)

5.40.1 Function Documentation

5.40.1.1 int main (int *argc*, char ** *argv*)

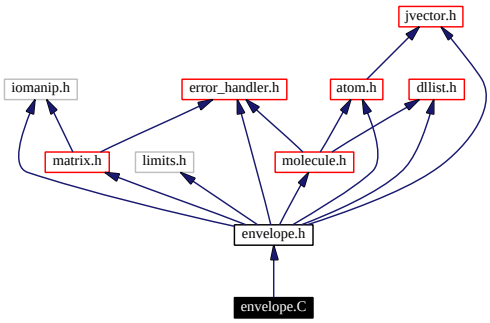
Definition at line 6 of file env_diff.C.

```
00006                                {
00007
00008   if(argc != 3)
00009     {
00010       cerr << "Usage: env-diff file file\n";
00011       return 0;
00012     }
00013
```

5.41 envelope.C File Reference

```
#include "envelope.h"
```

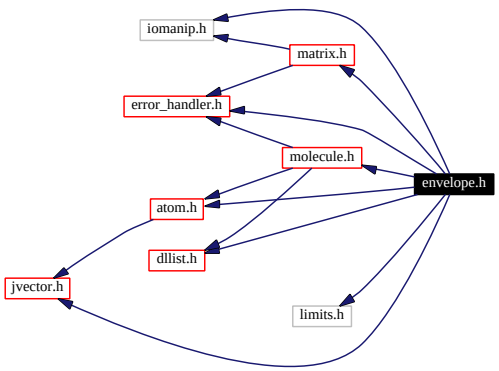
Include dependency graph for envelope.C:



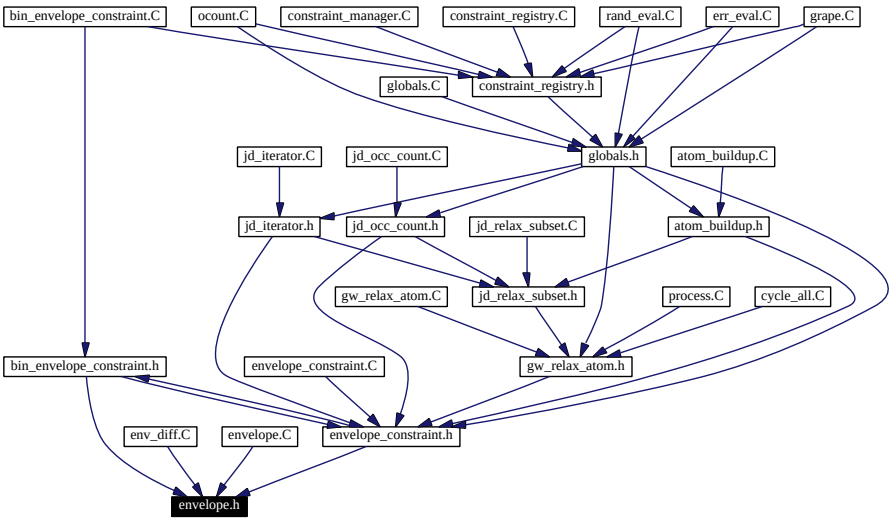
5.42 envelope.h File Reference

```
#include <limits.h>
#include <iomanip.h>
#include "atom.h"
#include "molecule.h"
#include "jvector.h"
#include "dllist.h"
#include "matrix.h"
#include "error_handler.h"
```

Include dependency graph for envelope.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Envelope**

Defines

- #define **MAX_ENV_DEN** 1.
- #define **MIN_ENV_DEN** 0.
- #define **ENV_DEN_RANGE** 1.
- #define **DEFAULT_CUBE_WIDTH** 3.7

5.42.1 Define Documentation

5.42.1.1 #define DEFAULT_CUBE_WIDTH 3.7

Definition at line 29 of file envelope.h.

5.42.1.2 #define ENV_DEN_RANGE 1.

Definition at line 17 of file envelope.h.

5.42.1.3 #define MAX_ENV_DEN 1.

//

envelope.h

This is a surface envelope class. It holds the points and data for a molucular envelope.

//

Definition at line 15 of file envelope.h.

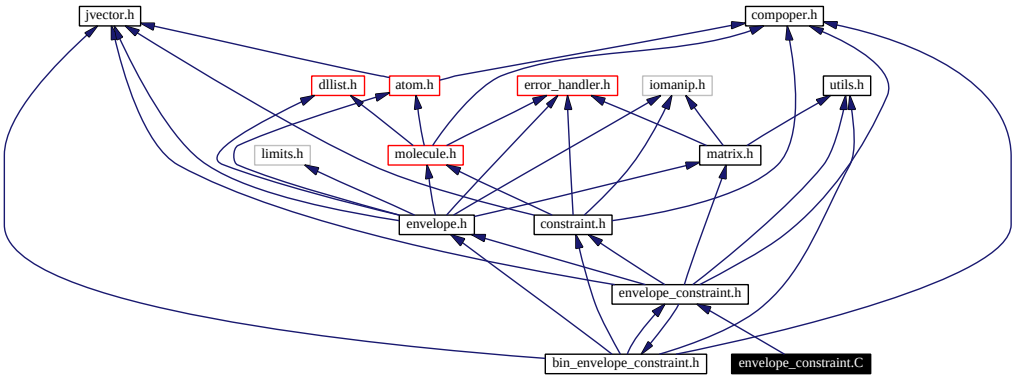
5.42.1.4 #define MIN_ENV_DEN 0.

Definition at line 16 of file envelope.h.

5.43 envelope_constraint.C File Reference

```
#include "envelope_constraint.h"
```

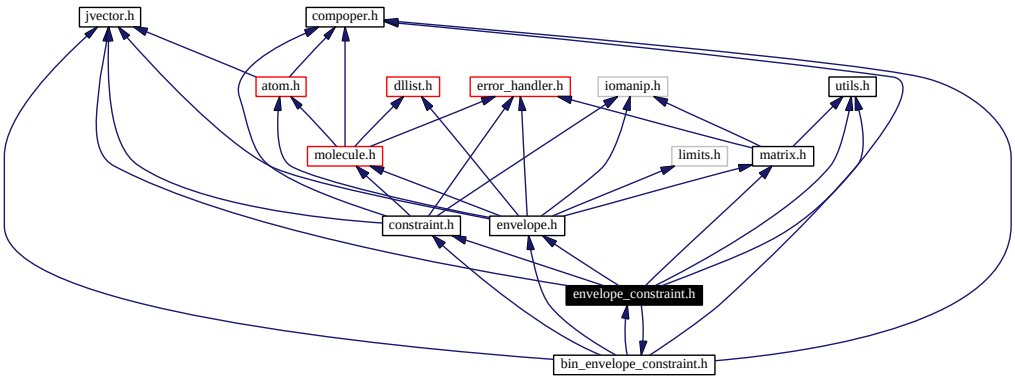
Include dependency graph for envelope_constraint.C:



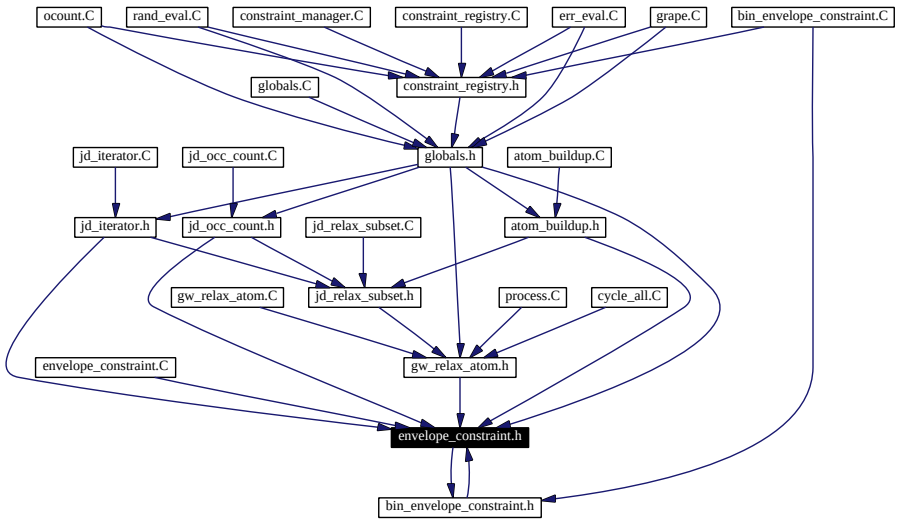
5.44 envelope_constraint.h File Reference

```
#include "envelope.h"
#include "bin_envelope_constraint.h"
#include "constraint.h"
#include "compoper.h"
#include "jvector.h"
#include "matrix.h"
#include "utils.h"
```

Include dependency graph for envelope_constraint.h:



This graph shows which files directly or indirectly include this file:



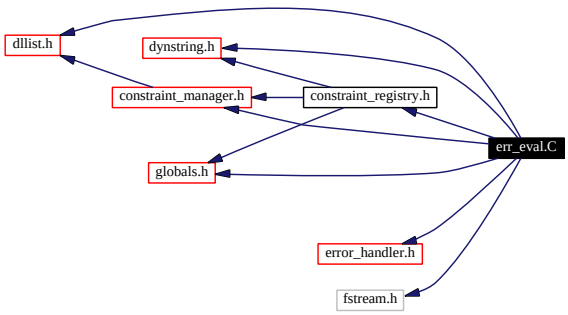
Compounds

- class **EnvelopeConstraint**

5.45 err_eval.C File Reference

```
#include <fstream.h>
#include "dynstring.h"
#include "dllist.h"
#include "constraint_manager.h"
#include "constraint_registry.h"
#include "globals.h"
#include "error_handler.h"
```

Include dependency graph for err_eval.C:



Functions

- void **print_usage** (ostream &os)

  Prints information about usage.

- void **print_help** (ostream &os)

  Prints full help.

- int **main** (int argc, char **argv)

5.45.1 Function Documentation

5.45.1.1 int main (int *argc*, char ** *argv*)

Definition at line 17 of file err_eval.C.

```
00017                                {
00018   if(argc < 2)
00019     {
00020       print_usage(cout);
00021       return 1;
00022     }
00023
00024   cerr << "Begin: ";
00025   for (int i =0; i < argc; i++){
00026     cerr << argv[i]<<" ";
00027   }
```

```
00028   cerr << endl;
00029
00031   DLList<DynString> ArgList;
00032
00034   DLList<DynString> ConstraintFileList;
00035
00037   Molecule * predictedMolecule=0;
00038
00040   DynString arg;
00041
00042   DynString positionFile = "";
00043
00045   ifstream * XmolInputFile=0;
00046
00047
00049   ifstream * input=0;
00050
00052   int argStart = 1;
00053
00055   DynString BaseFN = "evalenv-output";
00056
00057   int skip = 1;
00058
00059   // See if the user has specified a parameter file.  It is a
00060   // parameter file if the first argument does *not* start with
00061   // a dash, '-' (i.e. it is not a flag).
00062   //
00063
00064   if(argv[1][0] != '-')
00065     {
00066       // Read in the arguments from the parameter file.  This file should
00067       // contain flags (including their dashes) followed immediately by
00068       // the values associated with them.  Warning: this processing step
00069       // ignores white space!
00070       //
00071       if(!(input = new ifstream(argv[1])))
00072         {
00073           cerr << "Could not open/find parameter file, " << argv[1]
00074                << ", quitting. Try -help option.\n";
00075           return 2;
00076         }
00077       while(!input->eof() && !input->bad())
00078         {
00079           *input >> arg;
00080
00081           if(arg[0] == '#')
00082             arg.EatLine(*input);
00083           // Ignore lines that begin with a pound, '#', sign:
00084           else
00085             ArgList.AddToBack(arg);
00086         }
00087       ArgList.RemoveFromBack();  // Get rid of the eof character.
00088
00089       delete input; input=0;
00090       argStart=2;
00091     }
00092
00093   // Copy the arguments from the command line onto our argument list:
00094   //
00095   for(int a=argStart; a<argc; ++a)
00096     ArgList.AddToBack(DynString(argv[a]));
00097
00098   while(ArgList)
00099     {
00100       arg = ArgList.RemoveFromFront();
```

```
00101
00102       if((arg == DynString("-residues")) ||
00103               (arg == DynString("-res"))) {
00104
00105         // Check to see if we've already read in a molecule:
00106           if(predictedMolecule)
00107             {
00108               cerr << "Warning: More than one file has been supplied for "
00109                    << "the initial molecule.\nWill use the molecule "
00110                    << "specified in the file listed last.\n";
00111               delete predictedMolecule; predictedMolecule=0;
00112             }
00113           arg = ArgList.RemoveFromFront();  // Get number of atoms
00114           if((input = new ifstream(arg.GetData())))
00115             {
00116               predictedMolecule = new Molecule;
00117
00118               if(predictedMolecule->readFasta(*input)<2)
00119                 {
00120                   cerr << "Error generating molecule from residue sequence.\n";
00121                   delete predictedMolecule; predictedMolecule=0;
00122                 }
00123               delete input;input = 0;
00124             } else {
00125               cerr << "Could not open initial FASTA file, " << arg
00126                    << ", will try default location.\n";
00127
00128             }
00129       }
00130
00131       // Initial atom positions
00132       //
00133       else if((arg == DynString("-initial")) ||
00134               (arg == DynString("-init")) ||
00135               (arg == DynString("-i"))) {
00136
00137         positionFile = ArgList.RemoveFromFront();  // Get the name of the xmol file.
00138
00139       }
00140
00141
00142 //   while(ArgList)
00143 //     {
00144 //       arg = ArgList.RemoveFromFront();
00145 //
00146 //       // Initial molecule file is being specified.
00147 //       //
00148 //
00149 //       if((arg == DynString("-initial")) ||
00150 //       (arg == DynString("-init")) ||
00151 //       (arg == DynString("-i")))
00152 //      {
00153 //        // Check to see if we've already read in a molecule:
00154 //        if(predictedMolecule)
00155 //          {
00156 //            cerr << "Warning: More than one file has been supplied for "
00157 //                 << "the initial molecule.\nWill use the molecule "
00158 //                 << "specified in the file listed last.\n";
00159 //            delete predictedMolecule; predictedMolecule=0;
00160 //          }
00161 //
00162 //        arg = ArgList.RemoveFromFront();  // Get the name of the xmol file.
00163 //        if((XmolInputFile = new ifstream(arg.GetData())))
00164 //          {
00165 //            predictedMolecule = new Molecule;
```

```
00166 //            if(predictedMolecule->readXmol(*XmolInputFile)<1)
00167 //              {
00168 //                cerr << "Error reading initial molecule.\n";
00169 //                delete predictedMolecule; predictedMolecule=0;
00170 //              }
00171 //          }
00172 //        else
00173 //          {
00174 //            cerr << "Could not open initial molecule file, " << arg
00175 //                 << ", will try default location.\n";
00176 //          }
00177 //      }
00178
00179       else if(arg == DynString("-f") ||
00180               arg == DynString("-filebase"))
00181         {
00182             arg = ArgList.RemoveFromFront();
00183             if (arg[0] != '-'){
00184               BaseFN = arg;
00185             } else {
00186               DIE("Confused by filename ");
00187             }
00188         }
00189
00190       else if(arg == DynString("-skip"))
00191         {
00192           arg = ArgList.RemoveFromFront();
00193           if (arg[0] != '-'){
00194             skip = arg.ToInt();
00195           } else {
00196             DIE("Confused by filename ");
00197           }
00198         }
00199
00200       else if((arg == DynString("-constraint")) ||
00201               (arg == DynString("-constraints")) ||
00202               (arg == DynString("-c")))
00203         {
00204           // Just add this file to the list of files that contain
00205           // constraints for right now.  We'll read these in once
00206           // we're sure we have an initial molecule.
00207           //
00208           arg = ArgList.RemoveFromFront();
00209           ConstraintFileList.AddToBack(arg);
00210         }
00211
00212
00213       else if(arg == DynString("-help"))
00214         {
00215           print_help(cout);
00216           return 1;
00217
00218         }
00219
00220       // Ignore the bogus parameter.
00221       //
00222       else
00223         {
00224           cerr << "Unknown argument, " << arg << ", given, ignoring.\n";
00225         }
00226     }
00227
00228   // If we havent read in an initial molecule, we'll check the default
00229   // location for one:
00230   //
```

```
00231   if(!predictedMolecule)
00232     {
00233       cerr << "No initial molecule specified (-residues), quitting. Try -help option.\n";
00234       return 3;
00235     }
00236
00237
00238   if (positionFile != "") {
00239
00240     if((XmolInputFile = new ifstream(positionFile.GetData())))
00241       {
00242         if(predictedMolecule->readXmol(*XmolInputFile)<2)
00243           {
00244             cerr << "Error reading initial positions from " << positionFile << ".  Check the file.\n";
00245             delete predictedMolecule; predictedMolecule=0;
00246             return 4;
00247           }
00248       } else    {
00249         cerr << "Could not open initial molecule position file: " << positionFile << ".\n";
00250         return 5;
00251       }
00252
00253   } else {
00254     cerr << " You must use -i to specify initial positions" << endl;
00255     return 6;
00256   }
00257
00258   // Now that we've got an actual molecule, we can create a constraint
00259   // manager and read in the constraints:
00260   //
00261
00262   ConstraintManager CM(*predictedMolecule);
00263
00264   //
00265   //  cerr << " WARNING WARNING WARNING -- perturbing inputs \n\n" << endl;
00266   //  CM.RandomPerturb(.2);
00267   //
00268
00269   int num_read = 0;
00270
00271   while(ConstraintFileList)
00272     {
00273       // Get the filename and then open the file and then read the
00274       // constraints in using the constraint manager.
00275       arg = ConstraintFileList.RemoveFromFront();
00276       input = new ifstream(arg.GetData());
00277       if(input->good())
00278         {
00279           // Use the registry's routine to read in constraints and
00280           // append them to the constraint manager.
00281           //
00282
00283           num_read += ConstraintRegistry::readConstraints(*input,CM);
00284           delete input; input=0;
00285         } else {
00286           cerr << "\n\nWhile reading file: " << arg.GetData() << endl;
00287           DIE ("Cannot open file");
00288         }
00289
00290     }
00291
00292   if (num_read == 0) {
00293     WARN ("You have no constraints");
00294   }
00295
```

```
00296
00297   jdErr_Eval myeval(CM, XmolInputFile, skip);
00298   myeval.BaseFN = BaseFN;
00299   myeval.run();
00300
00301   if (predictedMolecule)
00302     delete predictedMolecule;
00303   if (XmolInputFile)
00304     {delete XmolInputFile;XmolInputFile =0;}
00305
```

5.45.1.2 void print_help (ostream & *os*)

Prints full help.

Definition at line 321 of file err_eval.C.

```
00322 {
00323   os    << "\nHELP\n\n";
00324   print_usage(os);
00325   os    << "Additional Options:\n"
00326         << "  \n"
00327         << "  \n";
00328
00329
```

5.45.1.3 void print_usage (ostream & *os*)

Prints information about usage.

Definition at line 310 of file err_eval.C.

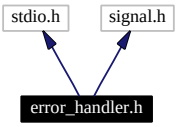
```
00311 {
00312   os    << "\nUsage: evalenv <-help> [optionfilename | options] \n\n"
00313         << "Required options:\n"
00314         << "  {-initial | -init | -i} <filename> }\n"
00315         << "     Initial molecule: xmol file\n"
00316         << "  {-constraints | -c}
00317
```

5.46 error_handler.h File Reference

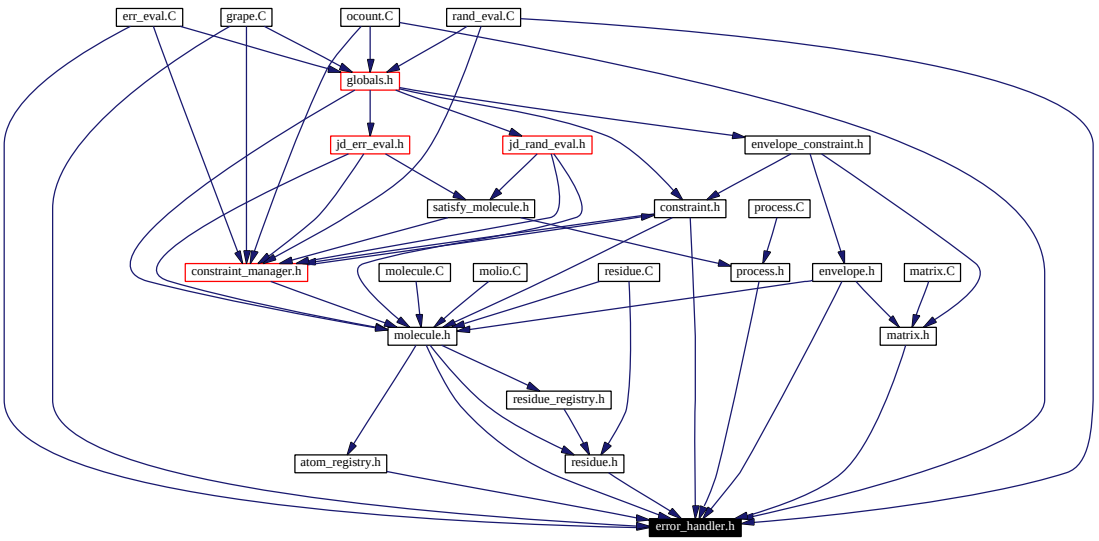
`#include <stdio.h>`

`#include <signal.h>`

Include dependency graph for error_handler.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **DIE**(TEXT)
- #define **WARN**(TEXT)
- #define **WARN_IF**(EXP, TEXT)

5.46.1 Define Documentation

5.46.1.1 #define DIE(TEXT)

Value:

```
do { \
fprintf (stderr, "\n\nERROR: " #TEXT "\n"); \
fprintf (stderr, "+%d %s\n\n", __LINE__, __FILE__); \
raise (SIGQUIT); \
} while (0)
```

Definition at line 7 of file error_handler.h.

Referenced by AnimFrame::AnimFrame(), AtomBuildup::AtomBuildup(), Envelope::Blur(), Envelope::CalcGradient(), EnvelopeConstraint::Compare(), Envelope::Compare(), BinEnvelopeConstraint::Compare(), EnvelopeConstraint::PreSetup(), Envelope::ReadAvsTxt(), Molecule::Reallocate(), RelaxSubset::RelaxSubset(), AnimFrame::Set_Output_Stream(), StopEval::StopEval(), ConstraintManager::TurnAtomOn(), Envelope::Upsize(), ConstraintManager::add(), Process::assertOrder(), Constraint::assertValidity(), PositionConstraint::computeStress(), EnvelopeConstraint::computeStress(), DistanceConstraint::computeStress(), BinEnvelopeConstraint::computeStress(), AtomConstraint::computeStress(), PositionConstraint::dStress(), DistanceConstraint::dStress(), AtomConstraint::dStress(), Matrix::det(), Matrix::findEigenVectors(), ConstraintRegistry::getConstraint(), ConstraintRegistry::getIntType(), ConstraintManager::getParents(), Matrix::identity(), jdErr_Eval::jdErr_Eval(), jdIterator::jdIterator(), main(), Constraint::notifyOff(), AtomConstraint::notifyRemove(), AtomConstraint::notifyValid(), Matrix::operator *(), Matrix::operator()(), Envelope::operator()(), Matrix::operator+(), Matrix::operator-(), ConstraintRegistry::readConstraints(), Molecule::readXmol(), ConstraintManager::registerEnvelope(), ConstraintManager::setMolecule(), ConstraintManager::setPosition(), Molecule::skipXmol(), and Constraint::stress().

5.46.1.2 #define WARN(TEXT)

Value:

```
do { \
fprintf (stderr, "\n\nWarning: " #TEXT "\n +%d %s\n\n", __LINE__, __FILE__ ); \
} while (0)
```

Definition at line 14 of file error_handler.h.

Referenced by EnvelopeConstraint::AlignModeltoCOM(), Envelope::Center(), Molecule::Reallocate(), AtomRegistry::SetType(), and main().

5.46.1.3 #define WARN_IF(EXP, TEXT)

Value:

```
do { if (EXP) \
        fprintf (stderr, "\n\nWarning: " #EXP #TEXT "\n +%d %s\n\n", __LINE__,  __FILE__ ); \
} while (0)
```

Definition at line 19 of file error_handler.h.

Referenced by Envelope::Build().

5.47 globals.C File Reference

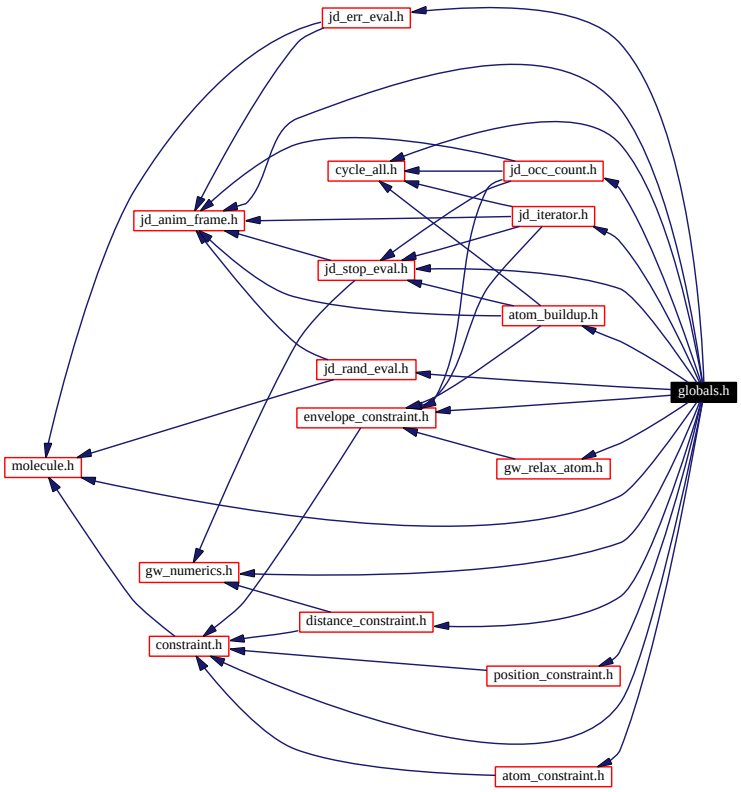
```
#include "globals.h"
```

Include dependency graph for globals.C:

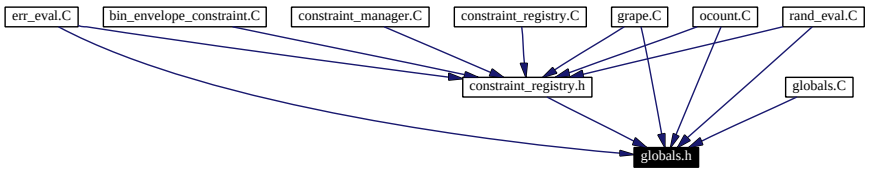
5.48 globals.h File Reference

```
#include "jd_iterator.h"
#include "gw_numerics.h"
#include "gw_relax_atom.h"
#include "cycle_all.h"
#include "jd_stop_eval.h"
#include "jd_anim_frame.h"
#include "jd_occ_count.h"
#include "jd_err_eval.h"
#include "jd_rand_eval.h"
#include "atom_buildup.h"
#include "molecule.h"
#include "constraint.h"
#include "atom_constraint.h"
#include "distance_constraint.h"
#include "envelope_constraint.h"
#include "position_constraint.h"
```

Include dependency graph for globals.h:



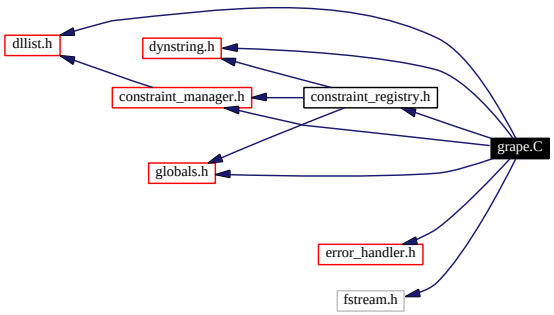
This graph shows which files directly or indirectly include this file:



5.49 grape.C File Reference

```
#include <fstream.h>
#include "dynstring.h"
#include "dllist.h"
#include "constraint_manager.h"
#include "constraint_registry.h"
#include "error_handler.h"
#include "globals.h"
```

Include dependency graph for grape.C:



Functions

- void **print_usage** (ostream &os)
  Prints information about usage.
- void **print_help** (ostream &os)
  Prints full help.
- int **main** (int argc, char **argv)

5.49.1 Function Documentation

5.49.1.1 int main (int *argc*, char ** *argv*)

Definition at line 17 of file grape.C.

```
00017                                {
00018   if(argc < 2)
00019     {
00020       print_usage(cout);
00021       return 0;
00022     }
00023
00024   cerr << "Begin: ";
00025   for (int i =0; i < argc; i++){
00026     cerr << argv[i]<<" ";
00027   }
```

```
00028   cerr << endl;
00029
00031   DLList<DynString> ArgList;
00032
00034   DLList<DynString> ConstraintFileList;
00035
00037   Molecule * predictedMolecule=0;
00038
00039   ofstream *AnimOut = 0;
00040   // default fstream for xmol frames
00041
00042   // Iterate through all the command line arguments:
00043   DynString arg;
00044
00045   // File for initial positions
00046   DynString positionFile = "";
00047
00048   SatisfyMolecule * TOP;
00049
00050   // Command-line argument.
00051   ifstream * input=0;
00052   // Parameter input stream.
00053   int argStart = 1;
00054
00055   // See if the user has specified a parameter file.  It is a
00056   // parameter file if the first argument does *not* start with
00057   // a dash, '-' (i.e. it is not a flag).
00058   //
00059
00060   DynString BaseFN = "grape-output"; // final xmol filename
00061   DynString AnimFN = ""; // anim filename -- defaults to off
00062
00063
00064   if(argv[1][0] != '-')
00065     {
00066       // Read in the arguments from the parameter file.  This file should
00067       // contain flags (including their dashes) followed immediately by
00068       // the values associated with them.  Warning: this processing step
00069       // ignores white space!
00070       //
00071       if(!(input = new ifstream(argv[1])))
00072         {
00073           cerr << "Could not open/find parameter file, " << argv[1]
00074                << ", quitting. Try -help option.\n";
00075           return 1;
00076         }
00077       while(!input->eof() && !input->bad())
00078         {
00079           *input >> arg;
00080
00081           if(arg[0] == '#')
00082             arg.EatLine(*input);
00083           // Ignore lines that begin with a pound, '#', sign:
00084           else
00085             ArgList.AddToBack(arg);
00086         }
00087       ArgList.RemoveFromBack();  // Get rid of the eof character.
00088
00089       delete input; input=0;
00090       argStart=2;
00091     }
00092
00093   // Copy the arguments from the command line onto our argument list:
00094   //
00095   for(int a=argStart; a<argc; ++a)
```

```
00096     ArgList.AddToBack(DynString(argv[a]));
00097
00098   while(ArgList)
00099     {
00100       arg = ArgList.RemoveFromFront();
00101
00102       if((arg == DynString("-residues")) ||
00103               (arg == DynString("-res"))) {
00104
00105         // Check to see if we've already read in a molecule:
00106           if(predictedMolecule)
00107             {
00108               cerr << "Warning: More than one file has been supplied for "
00109                    << "the initial molecule.\nWill use the molecule "
00110                    << "specified in the file listed last.\n";
00111               delete predictedMolecule; predictedMolecule=0;
00112             }
00113           arg = ArgList.RemoveFromFront();  // Get number of atoms
00114           if((input = new ifstream(arg.GetData())))
00115             {
00116               predictedMolecule = new Molecule;
00117
00118               if(predictedMolecule->readFasta(*input)<2)
00119                 {
00120                   cerr << "Error generating molecule from residue sequence.\n";
00121                   delete predictedMolecule; predictedMolecule=0;
00122                 }
00123               delete input;input = 0;
00124             } else {
00125               cerr << "Could not open initial FASTA file, " << arg
00126                    << ", will try default location.\n";
00127
00128             }
00129       }
00130
00131       // Initial atom positions
00132       //
00133       else if((arg == DynString("-initial")) ||
00134               (arg == DynString("-init")) ||
00135               (arg == DynString("-i"))) {
00136
00137         positionFile = ArgList.RemoveFromFront();  // Get the name of the xmol file.
00138
00139       }
00140
00141       else if(arg == DynString("-random")){
00142
00143         positionFile = "random";
00144
00145       }
00146
00147
00148       // Constraint file is being specified.
00149       //
00150       else if((arg == DynString("-constraint")) ||
00151               (arg == DynString("-constraints")) ||
00152               (arg == DynString("-c")))
00153         {
00154           // Just add this file to the list of files that contain
00155           // constraints for right now.  We'll read these in once
00156           // we're sure we have an initial molecule.
00157           //
00158           arg = ArgList.RemoveFromFront();
00159           ConstraintFileList.AddToBack(arg);
00160         }
```

```
00161
00162 //       // Read in van der Waal radius:
00163 //       //
00164 //       else if(arg == DynString("-vdw") ||
00165 //            arg == DynString("-vdwradius"))
00166 //      {
00167 //        double val[4];
00168 //        for(int i=0; i<4; ++i){
00169 //          if (ArgList.IsEmpty()) {
00170 //            DIE("Error while parsing the vdw arguments. Try -help option.");
00171 //          }
00172 //          arg = ArgList.RemoveFromFront();
00173 //          val[i] = arg.ToDbl();
00174 //        }
00175 //        gwNumerics::VDWRadius = val;
00176 //      }
00177 //
00178 //       // Read in local perturbation radius:
00179 //       //
00180 //       else if(arg == DynString("-lpr") ||
00181 //            arg == DynString("-lpradius"))
00182 //      {
00183 //        double val[4];
00184 //        for(int i=0; i<4; ++i){
00185 //          if (ArgList.IsEmpty()) {
00186 //            DIE("Error while  parsing the lpr arguments. Try -help option.");
00187 //          }
00188 //          arg = ArgList.RemoveFromFront();
00189 //          val[i] = arg.ToDbl();
00190 //        }
00191 //        gwNumerics::LPertRadius = val;
00192 //      }
00193
00194       // Set vdw checking vs all or vs moved
00195       //
00196       else if(arg == DynString("-vom"))
00197         {
00198           if (ArgList.IsEmpty()) {
00199             DIE("Error while  parsing the vom argument. Try -help option.");
00200           }
00201           arg = ArgList.RemoveFromFront();
00202           if(arg == DynString("true") || arg == DynString("t") ||
00203              arg == DynString("True") || arg == DynString("TRUE") ||
00204              arg == DynString("T") || arg == DynString("1"))
00205             gwNumerics::UseOnlyMoved = true;
00206           else if(arg == DynString("false") || arg == DynString("f") ||
00207                   arg == DynString("False") || arg == DynString("FALSE") ||
00208                   arg == DynString("F") || arg == DynString("0"))
00209             gwNumerics::UseOnlyMoved = false;
00210           else cerr << "Unable do understand vom arg. Not set -- defaults to true\n";
00211         }
00212
00213
00214
00215       else if(arg == DynString("-f") ||
00216               arg == DynString("-filebase"))
00217         {
00218             arg = ArgList.RemoveFromFront();
00219             if (arg[0] != '-'){
00220               BaseFN = arg;
00221             } else {
00222               DIE("Confused by filename ");
00223             }
00224         }
00225
```

```
00226
00227       // Read in flag for local perturbations:
00228       //
00229       else if(arg == DynString("-lp") ||
00230               arg == DynString("-lpert") )
00231         {
00232           arg = ArgList.RemoveFromFront();
00233           if(arg == DynString("true") || arg == DynString("t") ||
00234              arg == DynString("True") || arg == DynString("TRUE") ||
00235              arg == DynString("T") || arg == DynString("1"))
00236             gwNumerics::LocalPerturb = true;
00237           else if(arg == DynString("false") || arg == DynString("f") ||
00238                   arg == DynString("False") || arg == DynString("FALSE") ||
00239                   arg == DynString("F") || arg == DynString("0"))
00240             gwNumerics::LocalPerturb = false;
00241           else cerr << "Unable do understand lp or lpert. Not set.\n";
00242         }
00243
00244       // Least-squares success tolerance:
00245       //
00246       else if(arg == DynString("-lssucc"))
00247         {
00248           arg = ArgList.RemoveFromFront();
00249           gwNumerics::ls_succ_tolerance = arg.ToDbl();
00250         }
00251
00252       // Read in the least-squares derivative tolerance:
00253       //
00254       else if(arg == DynString("-lsderiv"))
00255         {
00256           arg = ArgList.RemoveFromFront();
00257           gwNumerics::ls_deriv_tolerance = arg.ToDbl();
00258         }
00259
00260       // Read in the least-squares distance tolerance:
00261       //
00262       else if(arg == DynString("-lsdist"))
00263         {
00264           arg = ArgList.RemoveFromFront();
00265           gwNumerics::ls_dist_tolerance = arg.ToDbl();
00266         }
00267
00268       // Set the maximum number of iterations allowed:
00269       //
00270       else if(arg == DynString("-iter"))
00271         {
00272           arg = ArgList.RemoveFromFront();
00273           gwRelaxAtom::max_iterations = arg.ToInt();
00274         }
00275
00276       // Set the number of cycles:
00277       //
00278       else if(arg == DynString("-cycles"))
00279         {
00280           arg = ArgList.RemoveFromFront();
00281           int n = arg.ToInt();
00282           jdIterator::MaxIterations = n;
00283           StopEval::Enable = false;  // turn off auto stop (early) features
00284         }
00285
00286       else if((arg == DynString("-buildcycles")) ||
00287               (arg == DynString("-bc")) )
00288         {
00289           arg = ArgList.RemoveFromFront();
00290           AtomBuildup::MaxIterations = arg.ToInt();
```

```
00291           arg = ArgList.RemoveFromFront();
00292           AtomBuildup::MaxFinalIterations = arg.ToInt();
00293         }
00294
00295
00296       else if(arg == DynString("-cyclelimit"))
00297         {
00298           arg = ArgList.RemoveFromFront();
00299           jdIterator::MaxIterations = arg.ToInt();
00300         }
00301
00302       else if ((arg == DynString("-quiet")) ||
00303                (arg == DynString("-q")))
00304         {
00305           StopEval::Quiet = true;
00306         }
00307
00308       else if (arg == DynString("-rtp"))
00309         {
00310           StopEval::Quiet = true;
00311           StopEval::VerboseOn = false;
00312           StopEval::Rtp  = true;
00313         }
00314
00315       else if((arg == DynString("-anim")) ||
00316                (arg == DynString("-a")))
00317         {
00318           arg = ArgList.RemoveFromFront();
00319           AnimFrame::Enabled = true;
00320           AnimFN = arg;
00321         }
00322       else if(arg == DynString("-help"))
00323         {
00324           print_help(cout);
00325           return 0;
00326
00327         }
00328
00329       // Ignore the bogus parameter.
00330       //
00331       else
00332         {
00333           cerr << "Unknown argument, " << arg << ", given, ignoring.\n";
00334         }
00335     }// end processing of arglist
00336
00337
00338   // If we havent read in a set of residues, quit.
00339   if(!predictedMolecule)
00340     {
00341       cerr << "No initial molecule specified, quitting. Try -help option.\n";
00342       return 3;
00343     }
00344
00345   if (positionFile != "") {
00346     // no atom buildup, specify positions
00347
00348     if (positionFile != "random") {
00349       if((input = new ifstream(positionFile.GetData())))
00350         {
00351           if(predictedMolecule->readXmol(*input)<2)
00352             {
00353               cerr << "Error reading initial positions from " << positionFile << ".  Check the file.\n";
00354               delete predictedMolecule; predictedMolecule=0;
00355               return 4;
```

```
00356             }
00357           delete input; input=0;
00358         }
00359       else
00360         {
00361           cerr << "Could not open initial molecule file, " << positionFile << ".\n";
00362           return 5;
00363         }
00364     } else {
00365       // set postions randomly
00366       if(predictedMolecule->setRandom()<2)      {
00367         cerr << "Error generating random molecule.\n";
00368         delete predictedMolecule; predictedMolecule=0;
00369         return 6;
00370       }
00371     }
00372 
00373   }
00374 
00375   // Now that we've got an actual molecule, we can create a constraint
00376   // manager and read in the constraints:
00377   //
00378   ConstraintManager CM(*predictedMolecule);
00379   //  CM.RandomPerturb(.1);
00380 
00381   int num_read = 0;
00382 
00383   while(ConstraintFileList)
00384     {
00385       // Get the filename and then open the file and then read the
00386       // constraints in using the constraint manager.
00387       arg = ConstraintFileList.RemoveFromFront();
00388       input = new ifstream(arg.GetData());
00389       if(input->good())
00390         {
00391           // Use the registry's routine to read in constraints and
00392           // append them to the constraint manager.
00393           //
00394 
00395           num_read += ConstraintRegistry::readConstraints(*input,CM);
00396           delete input; input=0;
00397         } else {
00398           cerr << "\n\nWhile reading file: " << arg.GetData() << endl;
00399           DIE ("Cannot open file");
00400         }
00401 
00402     }
00403 
00404   if (num_read == 0) {
00405     DIE ("You have no constraints");
00406   }
00407 
00408   //
00409   // Begin Sanity Checks
00410   //
00411 
00412   //    DLList<Atom> LA;
00413   //     Atom atom;
00414   //     for(int i=0; i<predictedMolecule->Number(); ++i)
00415   //       {
00416   //         LA = CM.getAssocAtoms(i);
00417   //    cout << "atom(" << i << "): ";
00418   //    while(LA)
00419   //      {
00420   //        atom = LA.RemoveFromFront();
```

```
00421   //        cout << atom.ID() << ' ';
00422   //      }
00423   //         cout << '\n';
00424   //       }
00425   //
00426   //     DLList<Constraint*>::ConstIterator it;
00427   //     Constraint * c;
00428   //     for(it = CM.getConstraintList().GetConstIterator(); it; ++it)
00429   //       {
00430   //         c = *it;
00431   //    cout << *c << '\n';
00432   //       }
00433   //
00434   //
00435   //
00436   // End Sanity Checks
00437   //
00438
00439   if (positionFile != "") {
00440     TOP = new jdIterator(CM);
00441   } else {
00442     TOP = new AtomBuildup(CM);
00443   }
00444
```

5.49.1.2 void print_help (ostream & *os*)

Prints full help.

Definition at line 484 of file grape.C.

```
00485 {
00486   os    << "\nHELP for using GRAPE\n\n";
00487   print_usage(os);
00488   os    << "Additional Options:\n"
00489         << "  {-vdw | -vdwradius} n1 n2 n3 n4\n"
00490         << "     Sets a global Van Der Waals Radius for Objective Function \n"
00491         << "     Values less than n1 to radius value n2 and to value n4 for \n"
00492         << "     OFV > n3+n1.\n"
00493         << "  {-lp | -lpert} {0|1|true|false}\n"
00494         << "     Enables or disables local pertubation.\n"
00495         << "  {-lpr | -lpradius} n1 n2 n3 n4\n"
00496         << "     Sets the global local perturbation radius as described above\n"
00497         << "  {-cycles n}\n"
00498         << "     Forces the simluation to run for n cycles.\n"
00499         << "  {-cyclelimit n}\n"
00500         << "     Limits the simluation to run for at most n cycles.\n"
00501         << "  {-a | -anim} cout | cerr | filename \n"
00502         << "     Write out xmol animation frames to stream or file.\n"
00503         << "  {-f | -filebase} filename \n"
00504         << "     Write final atom positions to filename\n"
00505         << "  {-q | -quiet} \n"
00506         << "     Run in quiet mode. \n"
00507         << "  {-rtp}
00508
```

5.49.1.3 void print_usage (ostream & *os*)

Prints information about usage.

Definition at line 471 of file grape.C.

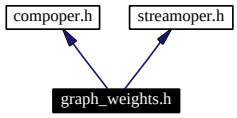
```
00472 {
00473   os    << "\nUsage: grape <-help> [optionfilename | options] \n\n"
00474         << "Required options:\n"
00475         << "  {-initial | -init | -i} <filename>  | -random n }\n"
00476         << "     Initial molecule: xmol file or random molecule with n atoms\n"
00477         << "  or {-residue | -res} <filename>}\n"
00478         << "     Initial molecule: fasta file, implies random start <for now>\n"
00479         << "  {-constraints | -c}
00480
```

5.50 graph_weights.h File Reference

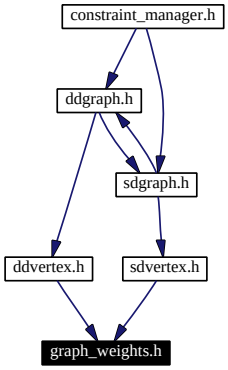
```
#include "compoper.h"
```

```
#include "streamoper.h"
```

Include dependency graph for graph_weights.h:



This graph shows which files directly or indirectly include this file:



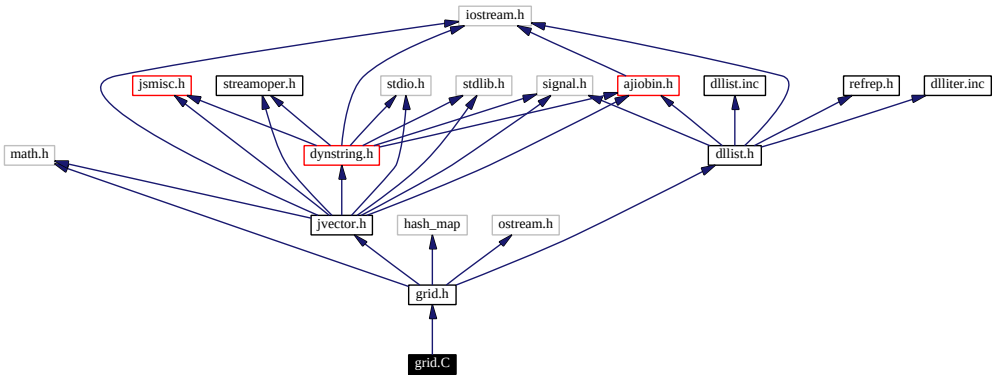
Compounds

- class **UnitWeight**

5.51 grid.C File Reference

```
#include "grid.h"
```

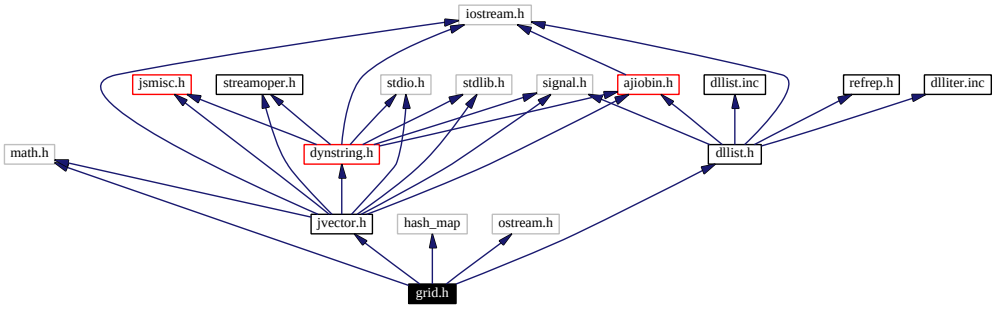
Include dependency graph for grid.C:



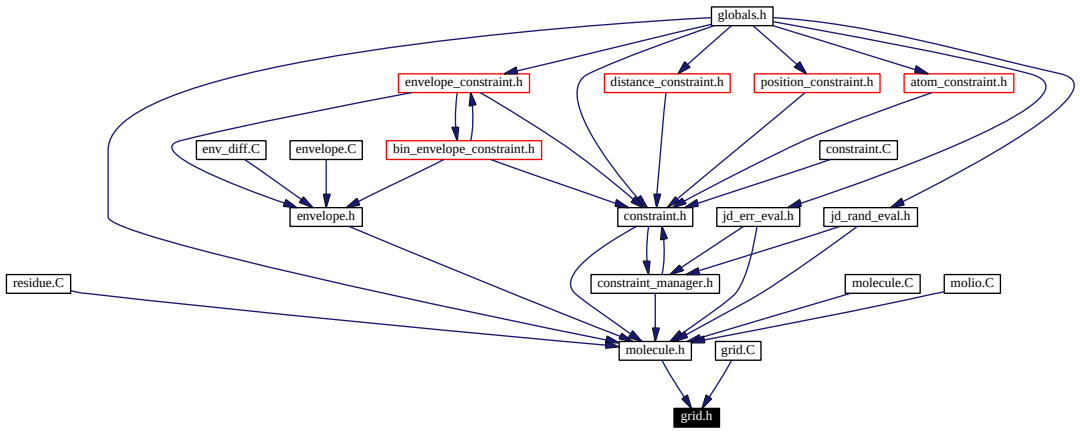
5.52 grid.h File Reference

```
#include <hash_map>
#include <ostream.h>
#include <math.h>
#include "jvector.h"
#include "dllist.h"
```

Include dependency graph for grid.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Grid**

Typedefs

- typedef long **GridNumber**
- typedef hash_multimap<const **GridNumber**, int, hash<long>, equal_to<long> > **hashmap_type**
- typedef **jVector*** **pointList**

5.52.1 Typedef Documentation

5.52.1.1 typedef long GridNumber

Definition at line 21 of file grid.h.

5.52.1.2 typedef hash_multimap< const GridNumber, int, hash< long >, equal_to< long > > hashmap_type

Definition at line 22 of file grid.h.

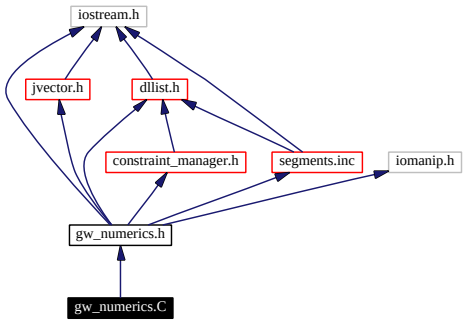
5.52.1.3 typedef jVector * pointList

Definition at line 23 of file grid.h.

5.53 gw_numerics.C File Reference

```
#include "gw_numerics.h"
```

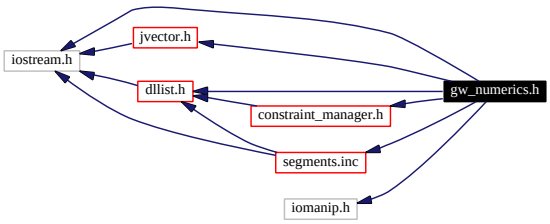
Include dependency graph for gw_numerics.C:



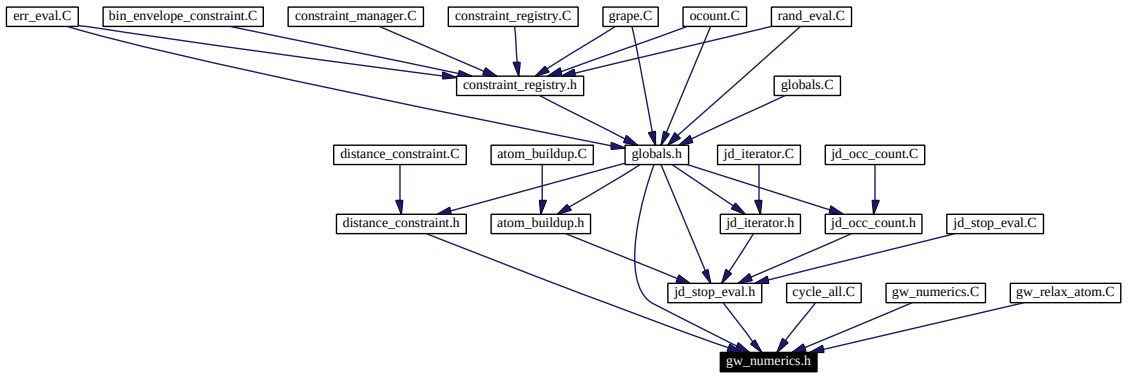
5.54 gw_numerics.h File Reference

```
#include <iostream.h>
#include <iomanip.h>
#include "jvector.h"
#include "constraint_manager.h"
#include "segments.inc"
#include "dllist.h"
```

Include dependency graph for gw_numerics.h:



This graph shows which files directly or indirectly include this file:



Compounds

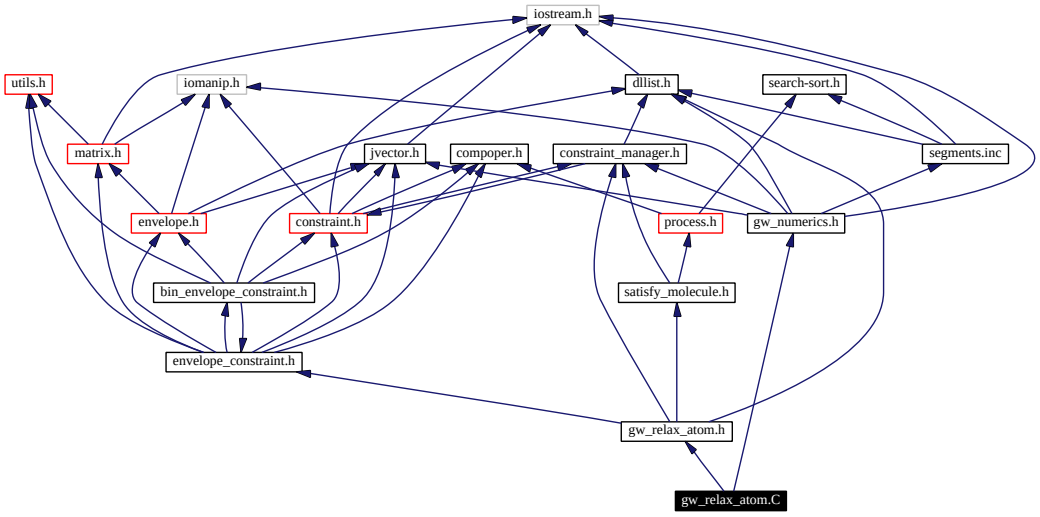
- class **gwNumerics**

5.55 gw_relax_atom.C File Reference

```
#include "gw_relax_atom.h"
```

```
#include "gw_numerics.h"
```

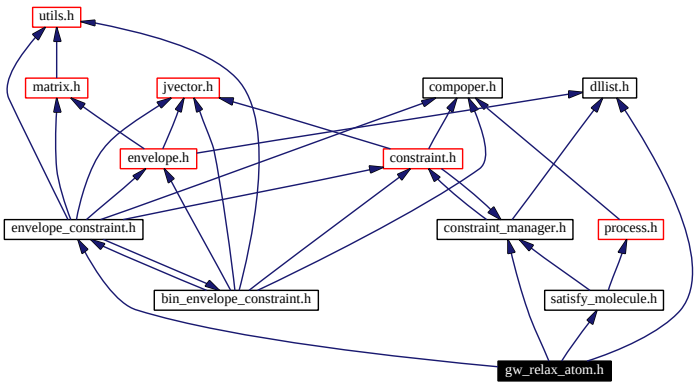
Include dependency graph for gw_relax_atom.C:



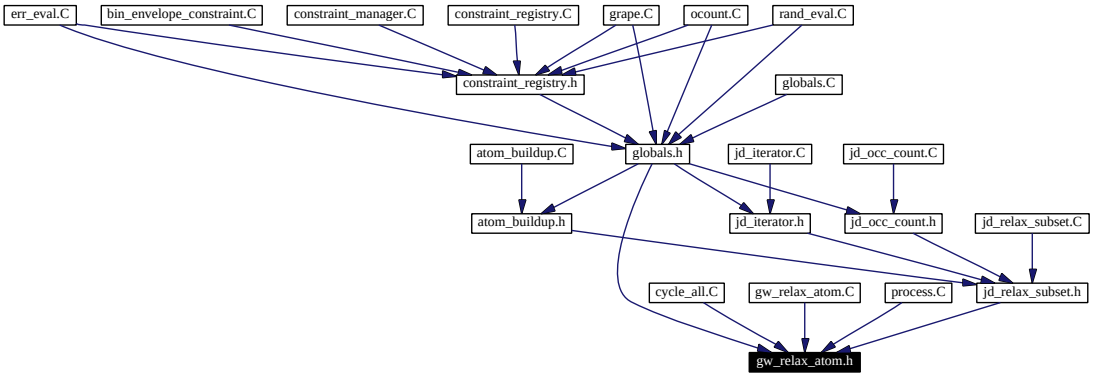
5.56 gw_relax_atom.h File Reference

```
#include "satisfy_molecule.h"
#include "constraint_manager.h"
#include "envelope_constraint.h"
#include "dllist.h"
```

Include dependency graph for gw_relax_atom.h:



This graph shows which files directly or indirectly include this file:



Compounds

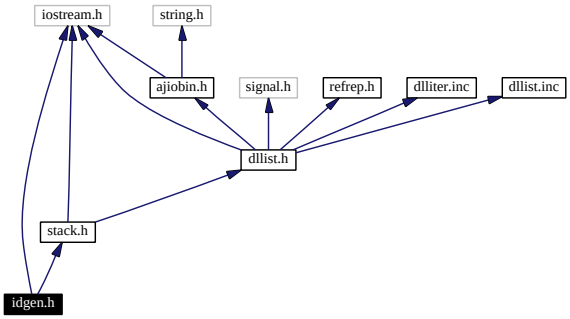
- class **gwRelaxAtom**

5.57 idgen.h File Reference

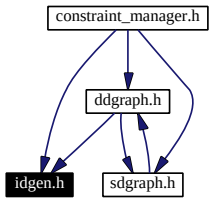
```
#include <iostream.h>
```

```
#include "stack.h"
```

Include dependency graph for idgen.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **IdGen**

Functions

- ostream& **operator**<< (ostream &os, const **IdGen** &idg)

5.57.1 Function Documentation

5.57.1.1 ostream & operator<< (ostream & *os*, const IdGen & *idg*) [inline]

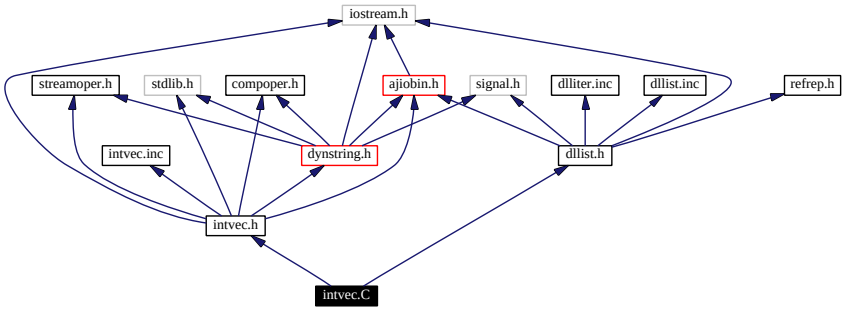
Definition at line 126 of file idgen.h.

```
00127   {
00128     os << '[' << idg.MaxId << "] {" << idg.FreeId << '}
00129
```

5.58 intvec.C File Reference

```
#include "intvec.h"
#include "dllist.h"
```

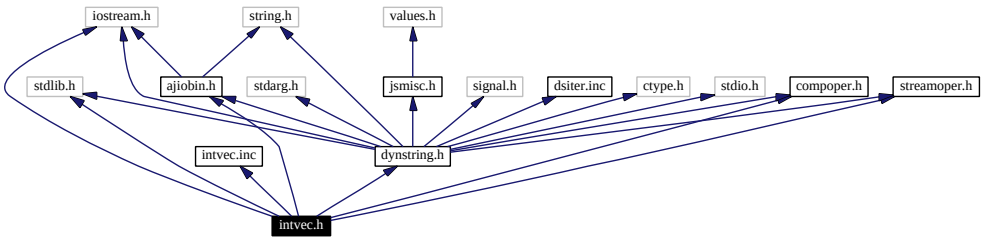
Include dependency graph for intvec.C:



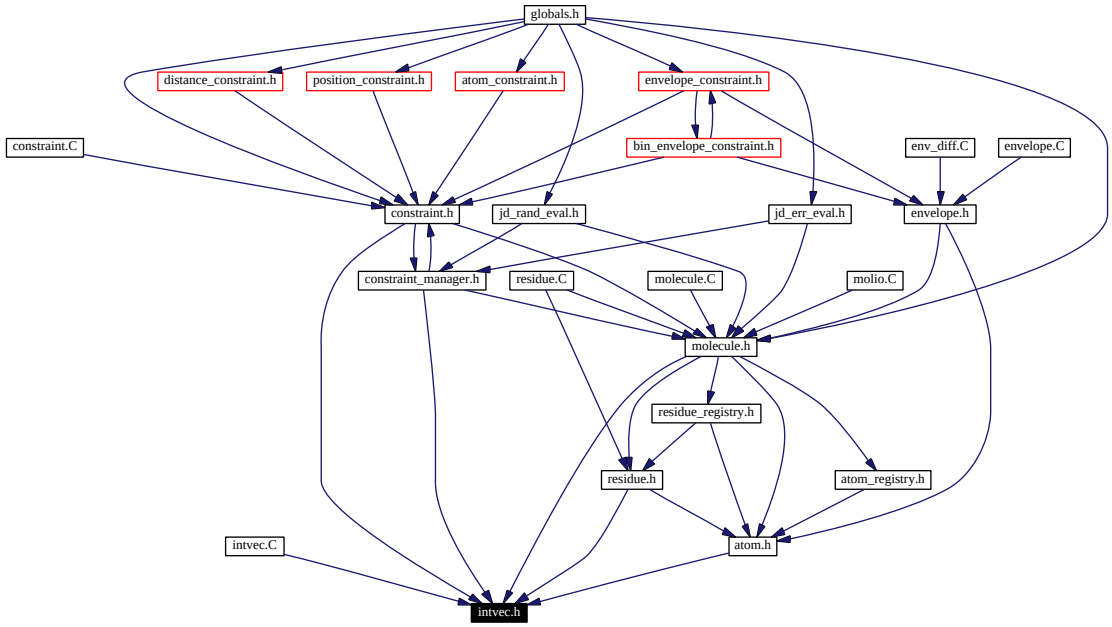
5.59 intvec.h File Reference

```
#include <iostream.h>
#include <stdlib.h>
#include "dynstring.h"
#include "ajiobin.h"
#include "compoper.h"
#include "streamoper.h"
#include <intvec.inc>
```

Include dependency graph for intvec.h:



This graph shows which files directly or indirectly include this file:

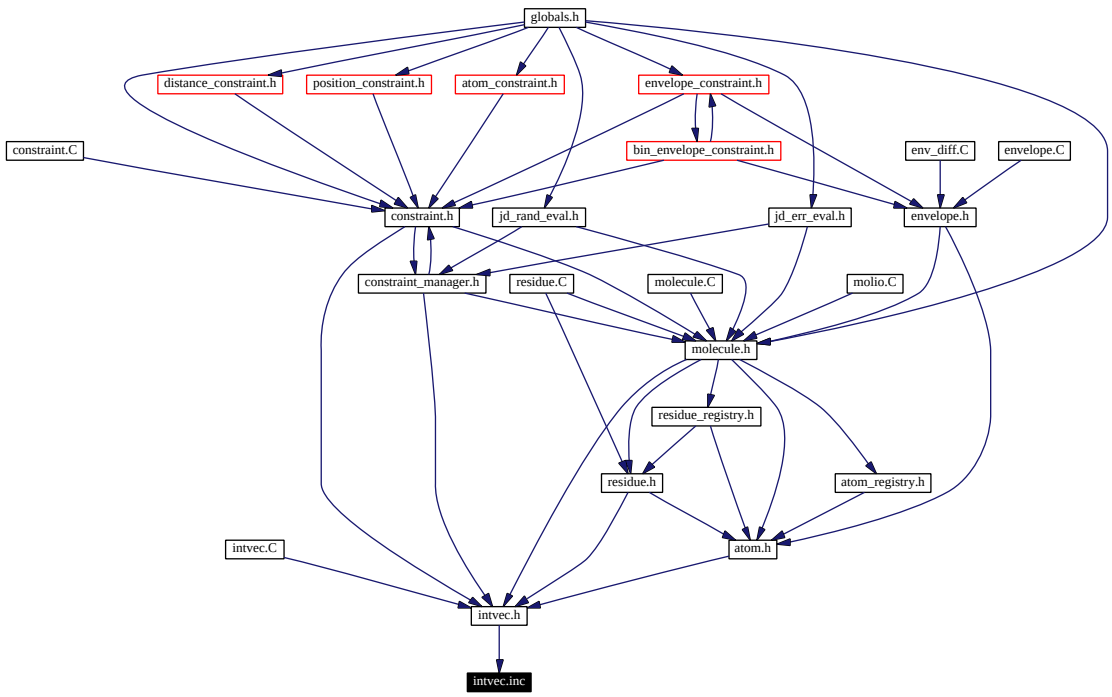


Compounds

- class **IntVec**

5.60 intvec.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- int **operator==** (const **IntVec** &a, const **IntVec** &b)
- int **operator!=** (const **IntVec** &a, const **IntVec** &b)
- int **operator<=** (const **IntVec** &a, const **IntVec** &b)
- int **operator>=** (const **IntVec** &a, const **IntVec** &b)
- int **operator<** (const **IntVec** &a, const **IntVec** &b)
- int **operator>** (const **IntVec** &a, const **IntVec** &b)
- **IntVec cat** (const **IntVec** &a, const **IntVec** &b)

5.60.1 Function Documentation

5.60.1.1 IntVec cat (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 233 of file intvec.inc.
Referenced by ConstraintManager::add().

```
00234   {
00235     IntVec c(a.L+b.L);
00236     for(int i=0; i<a.L; ++i) { c.I[i] = a.I[i]; }
00237     for(int i=0; i<b.L; ++i) { c.I[a.L+i] = b.I[i]; }
00238
```

5.60.1.2 int operator!= (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 127 of file intvec.inc.

```
00128   { return !(a == b); }
00129
```

5.60.1.3 int operator< (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 154 of file intvec.inc.

```
00155   {
00156     if(a.L > b.L) return 0;
00157     if(a.L < b.L) return 1;
00158     for(int i=0; i<a.L; ++i)
00159       {
00160         if(a.I[i] > b.I[i]) return 0;
00161         if(a.I[i] < b.I[i]) return 1;
00162       }
00163
```

5.60.1.4 int operator<= (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 130 of file intvec.inc.

```
00131   {
00132     if(a.L > b.L) return 0;
00133     if(a.L < b.L) return 1;
00134     for(int i=0; i<a.L; ++i)
00135       {
00136         if(a.I[i] > b.I[i]) return 0;
00137         if(a.I[i] < b.I[i]) return 1;
00138       }
00139
```

5.60.1.5 int operator== (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 119 of file intvec.inc.

```
00120   {
00121     if(a.L != b.L) return 0;
00122     for(int i=0; i<a.L; ++i)
00123         if(a.I[i] != b.I[i]) return 0;
00124     return 1;
00125
```

5.60.1.6 int operator> (const IntVec & *a*, const IntVec & *b*) [inline]

Definition at line 166 of file intvec.inc.

```
00167   {
00168     if(a.L < b.L) return 0;
00169     if(a.L > b.L) return 1;
00170     for(int i=0; i<a.L; ++i)
00171       {
```

```
00172         if(a.I[i] < b.I[i]) return 0;
00173         if(a.I[i] > b.I[i]) return 1;
00174       }
00175
```

5.60.1.7 int operator>= (const IntVec & *a*, const IntVec & *b*) [inline]

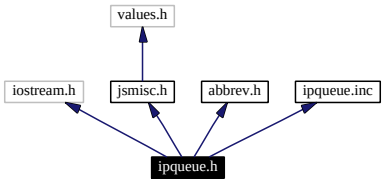
Definition at line 142 of file intvec.inc.

```
00143   {
00144     if(a.L < b.L) return 0;
00145     if(a.L > b.L) return 1;
00146     for(int i=0; i<a.L; ++i)
00147       {
00148         if(a.I[i] < b.I[i]) return 0;
00149         if(a.I[i] > b.I[i]) return 1;
00150       }
00151
```

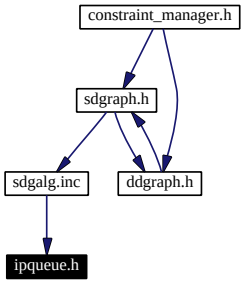
5.61 ipqueue.h File Reference

```
#include <iostream.h>
#include "jsmisc.h"
#include "abbrev.h"
#include <ipqueue.inc>
```

Include dependency graph for ipqueue.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **IntPriorityQueue**

Variables

- const int **IPQDEFLEN** = 1000

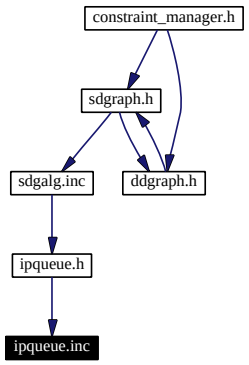
5.61.1 Variable Documentation

5.61.1.1 const int IPQDEFLEN = 1000

Definition at line 9 of file ipqueue.h.

5.62 ipqueue.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class PT> ostream& **operator**<< (ostream &os, const **IntPriorityQueue**< PT > &Q)

5.62.1 Function Documentation

5.62.1.1 template<class PT> ostream & operator<< (ostream & *os*, const IntPriorityQueue< PT > & *Q*) [inline]

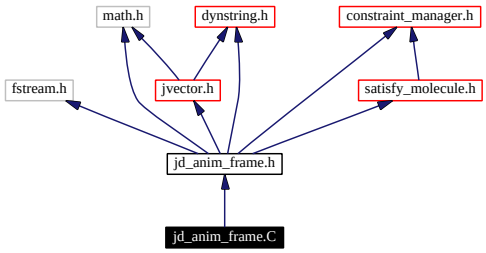
Definition at line 453 of file ipqueue.inc.

```
00454   {
00455     if(Q.IsBad())
00456       {
00457         os << "<<BADIPQUEUE>>";
00458         return os;
00459       }
00460
```

5.63 jd_anim_frame.C File Reference

```
#include "jd_anim_frame.h"
```

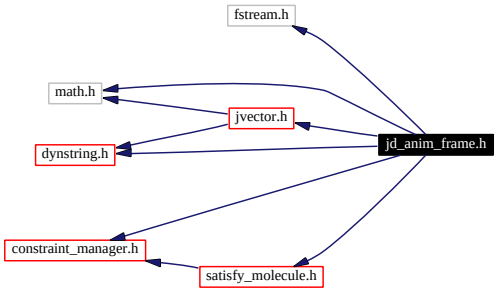
Include dependency graph for jd_anim_frame.C:



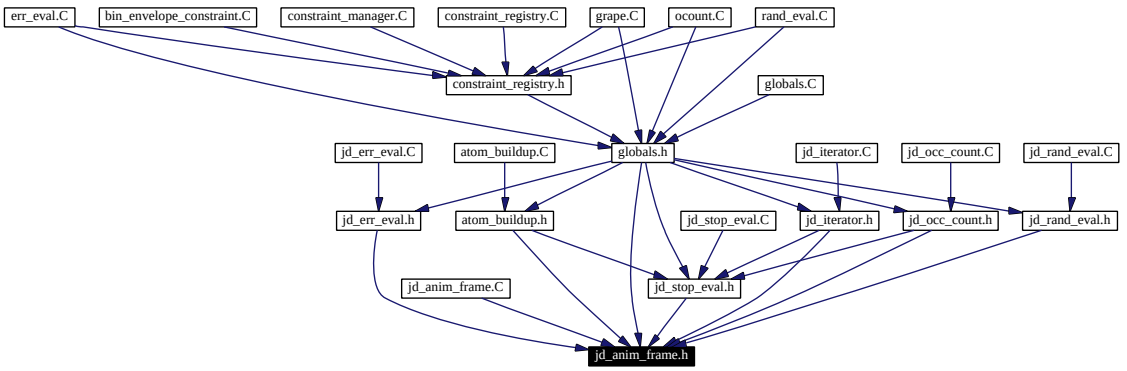
5.64 jd_anim_frame.h File Reference

```
#include <fstream.h>
#include "jvector.h"
#include "math.h"
#include "constraint_manager.h"
#include "satisfy_molecule.h"
#include "dynstring.h"
```

Include dependency graph for jd_anim_frame.h:



This graph shows which files directly or indirectly include this file:



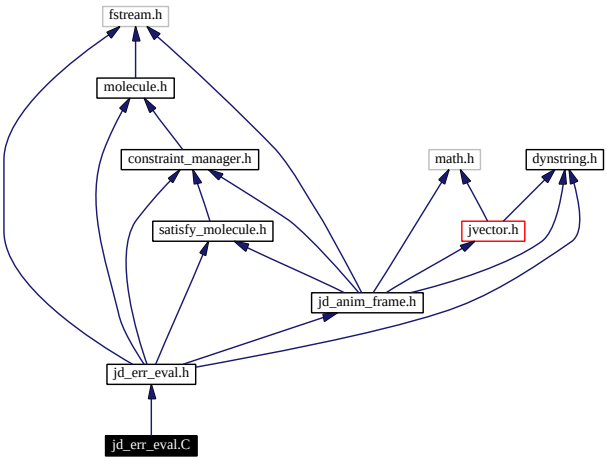
Compounds

- class **AnimFrame**

5.65 jd_err_eval.C File Reference

```
#include "jd_err_eval.h"
```

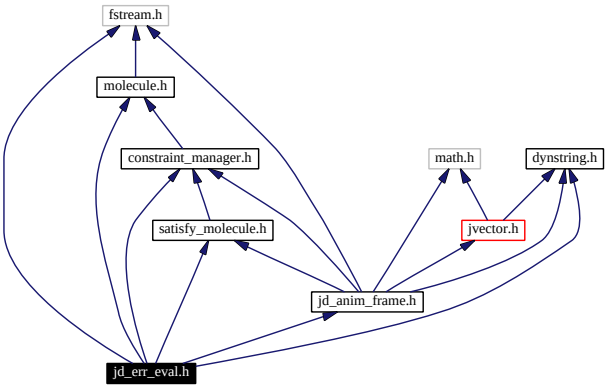
Include dependency graph for jd_err_eval.C:



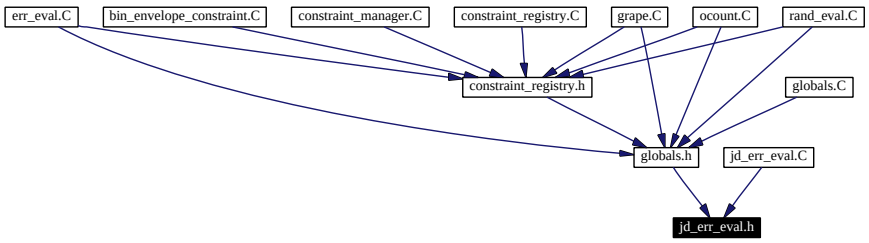
5.66 jd_err_eval.h File Reference

```
#include <fstream.h>
#include "jd_anim_frame.h"
#include "constraint_manager.h"
#include "satisfy_molecule.h"
#include "dynstring.h"
#include "molecule.h"
```

Include dependency graph for jd_err_eval.h:



This graph shows which files directly or indirectly include this file:



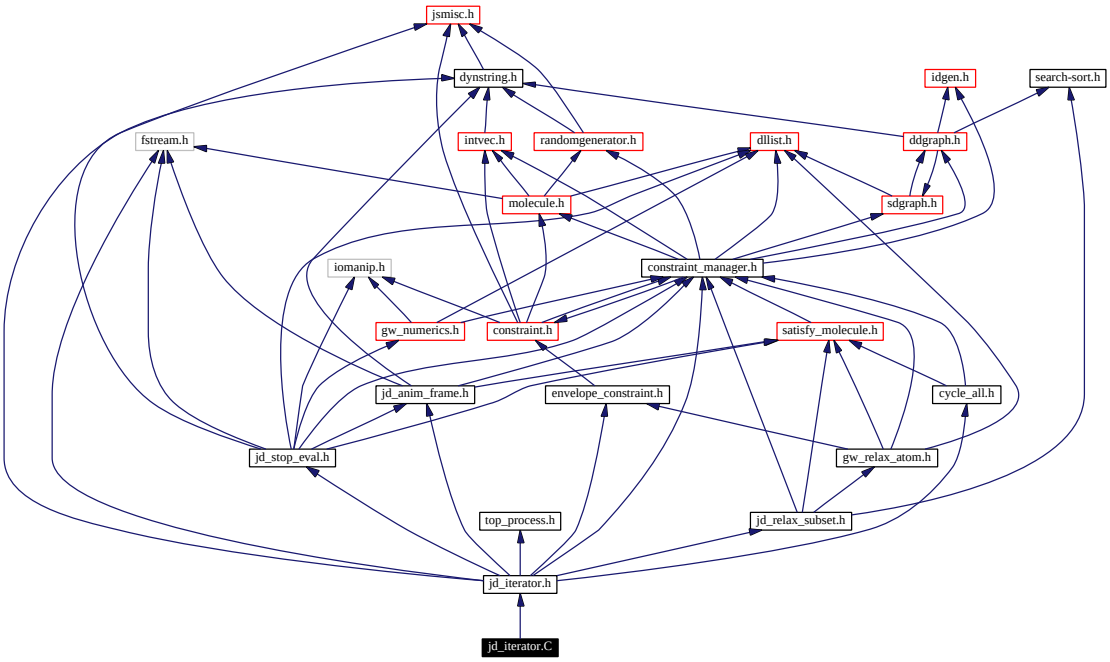
Compounds

- class **jdErr_Eval**

5.67 jd_iterator.C File Reference

```
#include "jd_iterator.h"
```

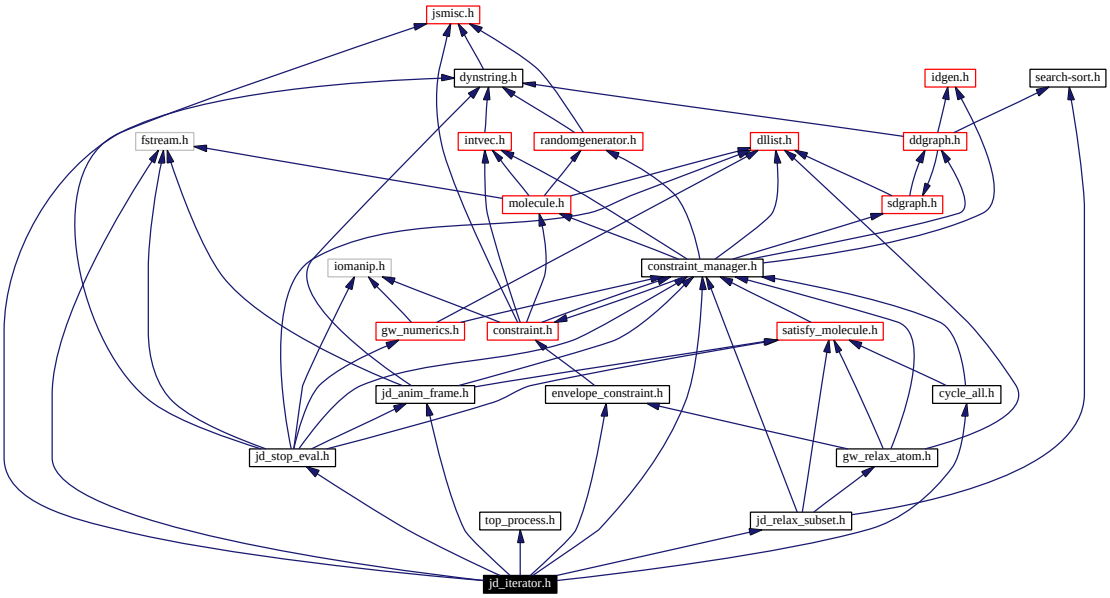
Include dependency graph for jd_iterator.C:



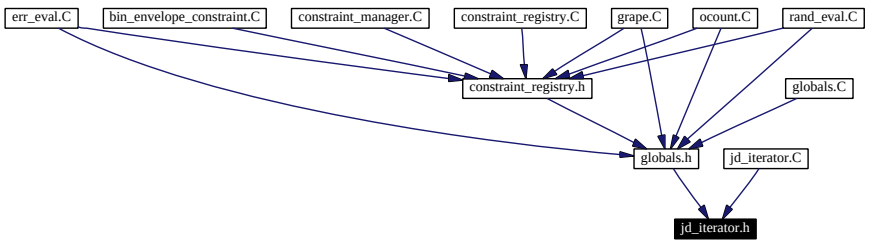
5.68 jd_iterator.h File Reference

```
#include <fstream.h>
#include "constraint_manager.h"
#include "top_process.h"
#include "dynstring.h"
#include "cycle_all.h"
#include "jd_stop_eval.h"
#include "jd_anim_frame.h"
#include "jd_relax_subset.h"
#include "envelope_constraint.h"
```

Include dependency graph for jd_iterator.h:



This graph shows which files directly or indirectly include this file:



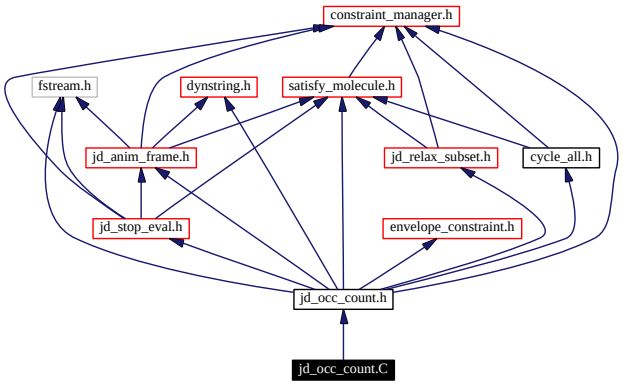
Compounds

- class **jdIterator**

5.69 jd_occ_count.C File Reference

```
#include "jd_occ_count.h"
```

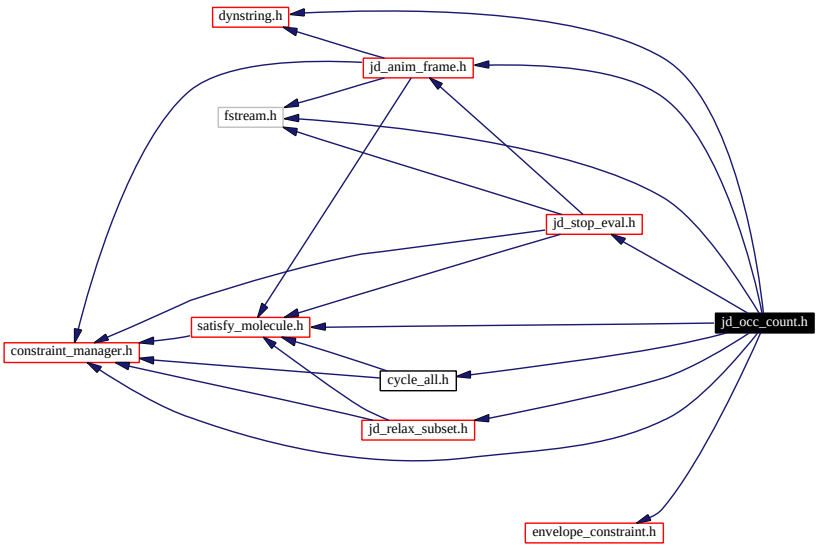
Include dependency graph for jd_occ_count.C:



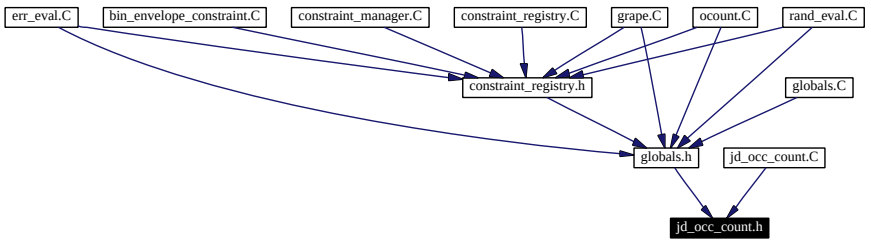
5.70 jd_occ_count.h File Reference

```
#include <fstream.h>
#include "constraint_manager.h"
#include "satisfy_molecule.h"
#include "dynstring.h"
#include "cycle_all.h"
#include "jd_stop_eval.h"
#include "jd_anim_frame.h"
#include "jd_relax_subset.h"
#include "envelope_constraint.h"
```

Include dependency graph for jd_occ_count.h:



This graph shows which files directly or indirectly include this file:



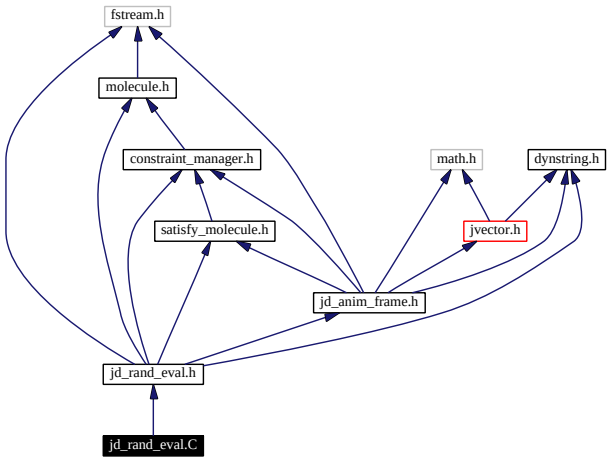
Compounds

- class **jdOcc_Count**

5.71 jd_rand_eval.C File Reference

```
#include "jd_rand_eval.h"
```

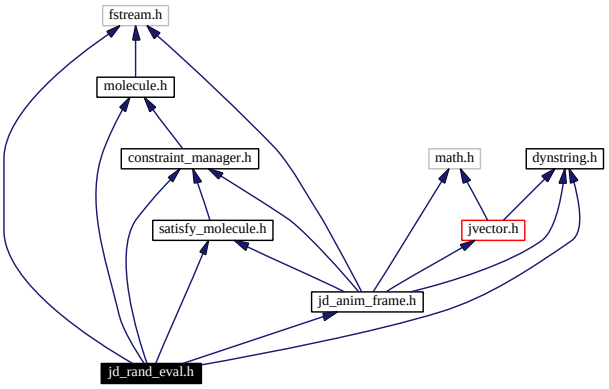
Include dependency graph for jd_rand_eval.C:



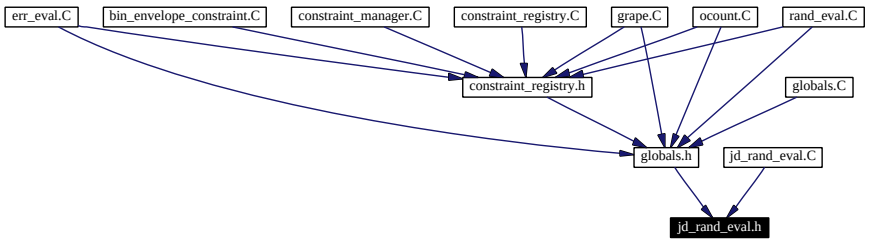
5.72 jd_rand_eval.h File Reference

```
#include <fstream.h>
#include "jd_anim_frame.h"
#include "constraint_manager.h"
#include "satisfy_molecule.h"
#include "dynstring.h"
#include "molecule.h"
```

Include dependency graph for jd_rand_eval.h:



This graph shows which files directly or indirectly include this file:



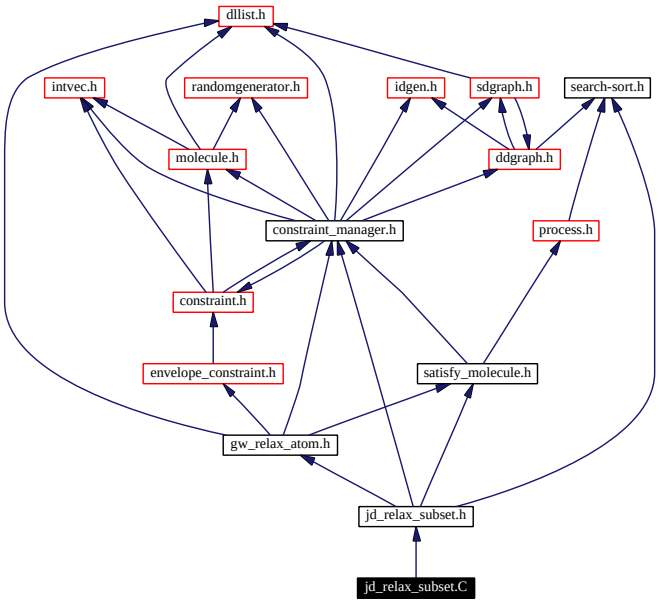
Compounds

- class **jdRand_Eval**

5.73 jd_relax_subset.C File Reference

```
#include "jd_relax_subset.h"
```

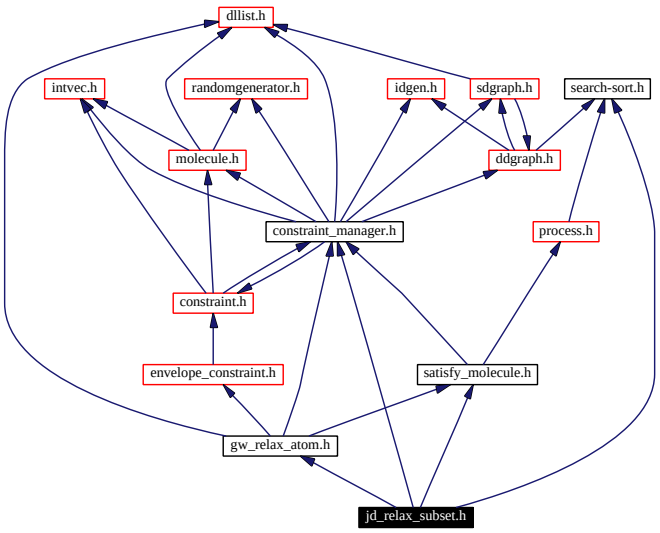
Include dependency graph for jd_relax_subset.C:



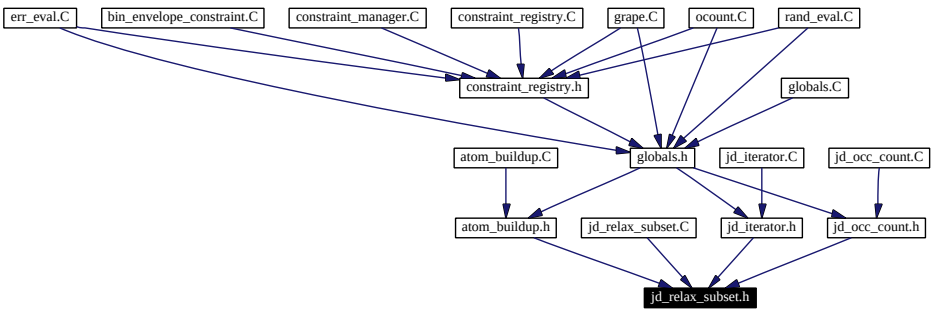
5.74 jd_relax_subset.h File Reference

```
#include "constraint_manager.h"
#include "search-sort.h"
#include "satisfy_molecule.h"
#include "gw_relax_atom.h"
```

Include dependency graph for jd_relax_subset.h:



This graph shows which files directly or indirectly include this file:



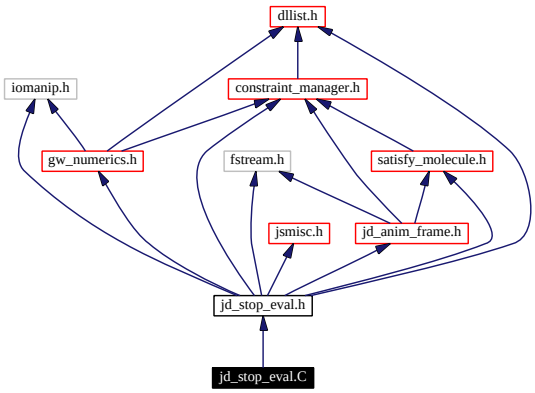
Compounds

- class **RelaxSubset**

5.75 jd_stop_eval.C File Reference

```
#include "jd_stop_eval.h"
```

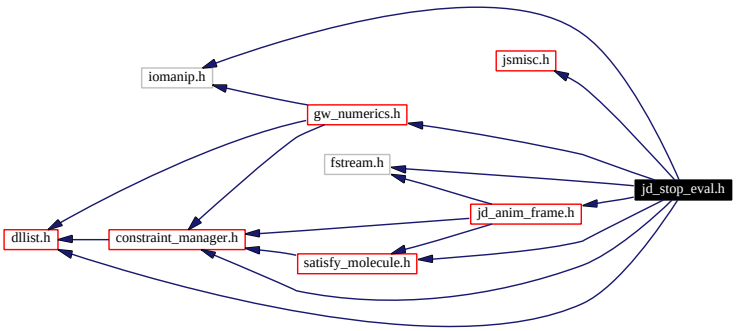
Include dependency graph for jd_stop_eval.C:



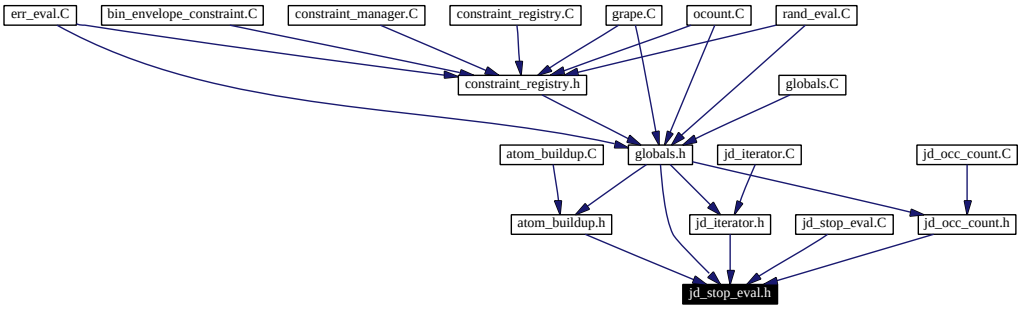
5.76 jd_stop_eval.h File Reference

```
#include <fstream.h>
#include <iomanip.h>
#include "constraint_manager.h"
#include "dllist.h"
#include "gw_numerics.h"
#include "satisfy_molecule.h"
#include "jsmisc.h"
#include "jd_anim_frame.h"
```

Include dependency graph for jd_stop_eval.h:



This graph shows which files directly or indirectly include this file:



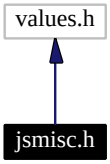
Compounds

- class **StopEval**

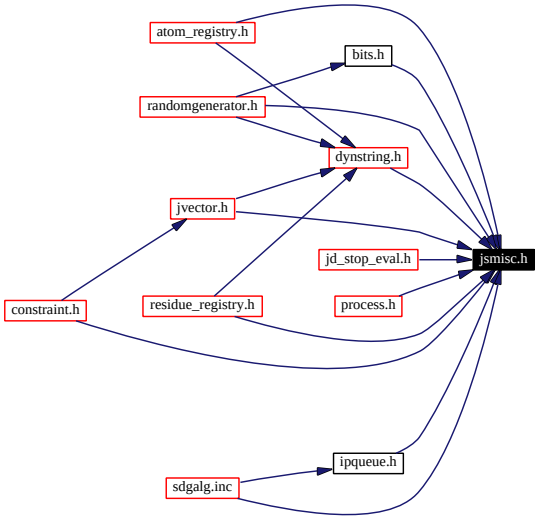
5.77 jsmisc.h File Reference

`#include <values.h>`

Include dependency graph for jsmisc.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **PI** 3.141592653589793116
- #define **INF** MAXDOUBLE∗MAXDOUBLE

Typedefs

- typedef unsigned int **uint**
- typedef short int **sint**
- typedef long int **lint**
- typedef unsigned short int **usint**
- typedef unsigned long int **ulint**
- typedef unsigned char **uchar**
- typedef class **ABitSet ABS**

5.77.1 Define Documentation

5.77.1.1 #define INF MAXDOUBLE*MAXDOUBLE

Definition at line 28 of file jsmisc.h.

5.77.1.2 #define PI 3.141592653589793116

Definition at line 24 of file jsmisc.h.

5.77.2 Typedef Documentation

5.77.2.1 typedef class ABitSet ABS

Definition at line 21 of file jsmisc.h.

Referenced by bf_iterator::bf_iterator(), df_iterator::df_iterator(), bf_iterator::operator=(), and df_iterator::operator=().

5.77.2.2 typedef long int lint

Definition at line 17 of file jsmisc.h.

5.77.2.3 typedef short int sint

Definition at line 16 of file jsmisc.h.

5.77.2.4 typedef unsigned char uchar

Definition at line 20 of file jsmisc.h.

5.77.2.5 typedef unsigned int uint

Definition at line 15 of file jsmisc.h.

5.77.2.6 typedef unsigned long int ulint

Definition at line 19 of file jsmisc.h.

5.77.2.7 typedef unsigned short int usint

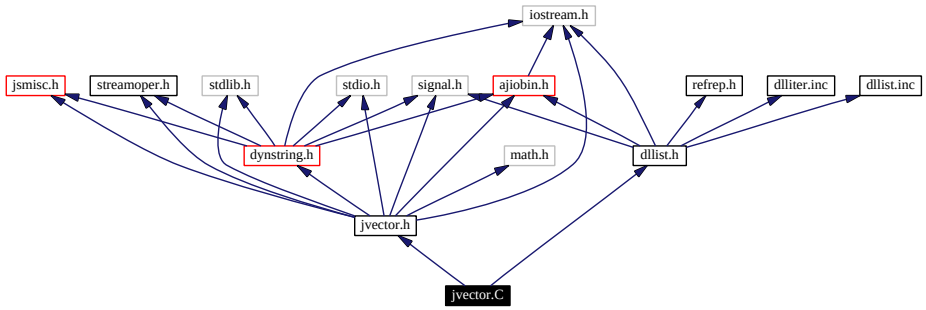
Definition at line 18 of file jsmisc.h.

5.78 jvector.C File Reference

```
#include "jvector.h"
```

```
#include "dllist.h"
```

Include dependency graph for jvector.C:



Functions

- **jVector operator-** (**jVector** x, const **jVector** &y)
- **jVector operator+** (**jVector** x, const **jVector** &y)
- double **operator** ∗ (**jVector** x, const **jVector** &y)
- **jVector operator%** (**jVector** x, const **jVector** &y)
- **jVector operator** ∗ (**jVector** y, double d)
- **jVector operator** ∗ (double d, **jVector** y)
- **jVector operator/** (**jVector** y, double d)
- **jVector operator%** (**jVector** x, double d)
- double **JVAngle** (const **jVector** &x, const **jVector** &y)
- double **euclid** (const **jVector** &x, const **jVector** &y)
- ostream& **VBPrintLine** (ostream &os, const **jVector** &x, const **jVector** &y, const **DynString** &color)
- ostream& **VBPrintPoint** (ostream &os, const **jVector** &x, const **DynString** &color)
- ostream& **VBPrintCircle** (ostream &os, const **jVector** &x, const double &radius, const **DynString** &color)
- double **Covariance** (const **jVector** &x, const **jVector** &y)

5.78.1 Function Documentation

5.78.1.1 double Covariance (const jVector & *x*, const jVector & *y*)

Definition at line 326 of file jvector.C.

```
00327   {
00328     if(x.L != y.L || x.L == 0)
00329         return INF;
00330
00331     double x_mean = x.mean();
00332     double y_mean = y.mean();
00333     double cov = 0.0;
00334
00335     for(int i=0; i<x.L; ++i)
```

```
00336         cov += (x[i] - x_mean)*(y[i] - y_mean);
00337     cov /= (x.L-1);
00338
00339     return cov;
00340
```

5.78.1.2 double JVAngle (const jVector & *x*, const jVector & *y*)

Definition at line 152 of file jvector.C.

```
00153   { return acos( (x*y) / x.mag() / y.mag() ); }
00154
```

5.78.1.3 ostream & VBPrintCircle (ostream & *os*, const jVector & *x*, const double & *radius*, const DynString & *color*)

Definition at line 264 of file jvector.C.

```
00266   {
00267
00268     if((x.L>3) || (x.L<2)) return os;
00269     char old_delim = jVector::IODelimiter;
00270     jVector::IODelimiter = ',';
00271
00272     if(x.L==3)
00273       os << "circle (" << x << ") " << radius << " :" << color;
00274     else if(x.L==2)
00275       os << "circle (" << x << ",0) " << radius << " :" << color;
00276     os << '\n';
00277     jVector::IODelimiter = old_delim;
00278     return os;
00279
```

5.78.1.4 ostream & VBPrintLine (ostream & *os*, const jVector & *x*, const jVector & *y*, const DynString & *color*)

Definition at line 226 of file jvector.C.

```
00228   {
00229     if((x.L != y.L) || (x.L>3) || (y.L>3) || (x.L<1) || (y.L<1)) return os;
00230
00231     char old_delim = jVector::IODelimiter;
00232     jVector::IODelimiter = ',';
00233
00234     if(x.L==3)
00235       os << "line (" << x << ") (" << y << ") :" << color;
00236     else if(x.L==2)
00237       os << "line (" << x << ",0) (" << y << ",0) :" << color;
00238     else if(x.L==1)
00239       os << "line (" << x << "0,0) (" << y << "0,0) :" << color;
00240     os << '\n';
00241     jVector::IODelimiter = old_delim;
00242     return os;
00243
```

5.78.1.5 ostream & VBPrintPoint (ostream & *os*, const jVector & *x*, const DynString & *color*)

Definition at line 245 of file jvector.C.

```
00246   {
00247
00248     if((x.L>3) || (x.L<1)) return os;
00249     char old_delim = jVector::IODelimiter;
00250     jVector::IODelimiter = ',';
00251
00252     if(x.L==3)
00253       os << "point " << x << " :" << color;
00254     else if(x.L==2)
00255       os << "point " << x << ",0 :" << color;
00256     else if(x.L==1)
00257       os << "point " << x << ",0,0 :" << color;
00258     os << '\n';
00259     jVector::IODelimiter = old_delim;
00260     return os;
00261
```

5.78.1.6 double euclid (const jVector & *x*, const jVector & *y*)

Definition at line 155 of file jvector.C.

Referenced by gwNumerics::addVDWSegments(), DistanceConstraint::computeAbsRes(), DistanceConstraint::computeStress(), PositionConstraint::dStress(), DistanceConstraint::dStress(), main(), and jdErr_Eval::run().

```
00156   {
00157     if(x.L != y.L) return -1.0;
00158     double sumproducts = 0.0;
00159     for(int i=0; i<x.L; ++i) sumproducts += pow(x.V[i]-y.V[i], 2.0);
00160     return sqrt(sumproducts);
00161
```

5.78.1.7 jVector operator * (double *d*, jVector *y*)

Definition at line 141 of file jvector.C.

```
00141                                         { return y*=d; }
00142
```

5.78.1.8 jVector operator * (jVector *y*, double *d*)

Definition at line 140 of file jvector.C.

```
00140                                         { return y*=d; }
00141
```

5.78.1.9 double operator * (jVector *x*, const jVector & *y*)

Definition at line 124 of file jvector.C.

```
00125 {
00126   double product_sum=0.0;
00127   if(x.L == y.L) for(int i=0; i<x.L; ++i) product_sum += x.V[i]*y.V[i];
00128   return product_sum;
00129
```

5.78.1.10 jVector operator% (jVector *x*, double *d*)

Definition at line 149 of file jvector.C.

```
00150   { return x%=d; }
00151
```

5.78.1.11 jVector operator% (jVector *x*, const jVector & *y*)

Definition at line 134 of file jvector.C.

```
00134                                                 { return x %= y; }
00135
```

5.78.1.12 jVector operator+ (jVector *x*, const jVector & *y*)

Definition at line 121 of file jvector.C.

```
00121                                                { return (x += y); }
00122
```

5.78.1.13 jVector operator- (jVector *x*, const jVector & *y*)

Definition at line 117 of file jvector.C.

```
00117                                                { return (x -= y); }
00118
```

5.78.1.14 jVector operator/ (jVector *y*, double *d*)

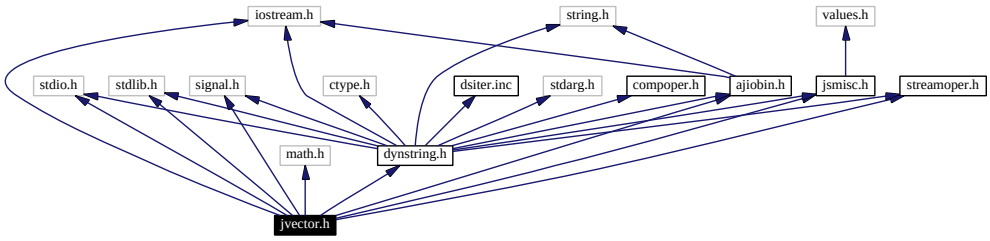
Definition at line 145 of file jvector.C.

```
00145                                          { return y/=d; }
00146
```

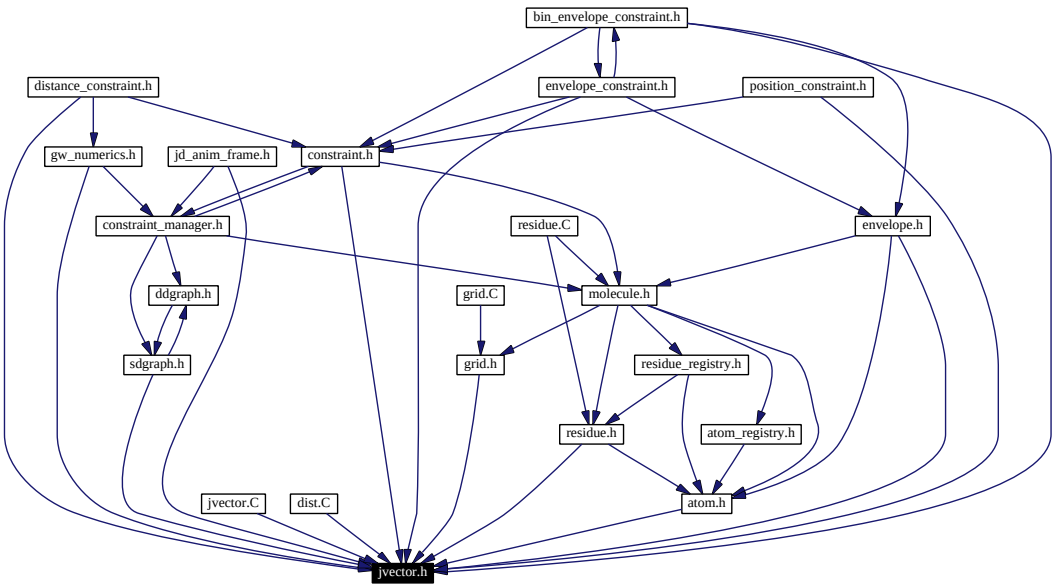
5.79 jvector.h File Reference

```
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <iostream.h>
#include <math.h>
#include "jsmisc.h"
#include "dynstring.h"
#include "ajiobin.h"
#include "streamoper.h"
```

Include dependency graph for jvector.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **jVector**

Functions

- int **doublecompare** (const void *x, const void *y)
- int **binsearch** (const **jVector** &v, double d)
- int **dcomp** (const void *x, const void *y)
- void **JVqsort** (**jVector** &v)

Variables

- const int **DEFAULTLEN** = 3

5.79.1 Function Documentation

5.79.1.1 void JVqsort (jVector & *v*) [inline]

Definition at line 329 of file jvector.h.

```
00329                         { qsort((void *) v.V, v.L, sizeof(double), dcomp); }
00330
```

5.79.1.2 int binsearch (const jVector & *v*, double *d*) [inline]

Definition at line 299 of file jvector.h.

```
00300 {
00301   // left index, right index, pivot, respectively.
00302   //
00303   int i=0;
00304   int j=v.L-1;
00305   int p;
00306
00307   while((i<=j) && (j>=0))
00308     {
00309       p = (j+i)/2;
00310       if(d < v.V[p]) j = p-1;
00311       else if(d > v.V[p]) i = p+1;
00312       else return p;
00313     }
00314
```

5.79.1.3 int dcomp (const void * *x*, const void * *y*) [inline]

Definition at line 319 of file jvector.h.

```
00320 {
00321   const double * X = (const double *) x;
00322   const double * Y = (const double *) y;
00323   if(*X < *Y) return -1;
00324   if(*X > *Y) return  1;
00325   return 0;
00326
```

5.79.1.4 int doublecompare (const void * x, const void * y)

5.79.2 Variable Documentation

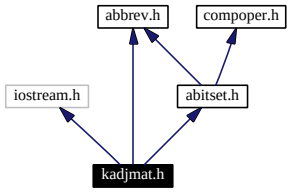
5.79.2.1 const int DEFAULTLEN = 3

Definition at line 25 of file jvector.h.

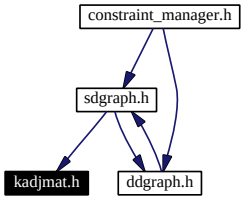
5.80 kadjmat.h File Reference

```
#include <iostream.h>
#include "abitset.h"
#include "abbrev.h"
```

Include dependency graph for kadjmat.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **KAdjMat**

5.81 Makefile File Reference

5.82 Makefile File Reference

5.83 Makefile File Reference

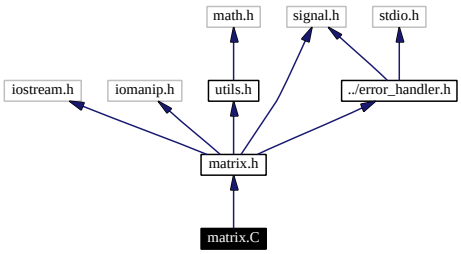
5.84 Makefile File Reference

5.85 Makefile File Reference

5.86 matrix.C File Reference

```
#include "matrix.h"
```

Include dependency graph for matrix.C:



Functions

- **Matrix operator** ∗ (double d, const **Matrix** &m)

5.86.1 Function Documentation

5.86.1.1 Matrix operator ∗ (double *d*, const Matrix & *m*)

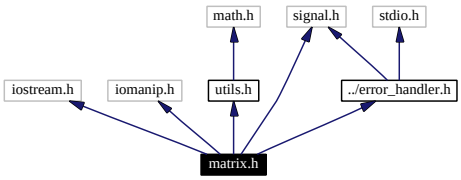
Definition at line 407 of file matrix.C.

```
00407                                          {
00408   int rows = m.getRows(), cols = m.getCols();
00409   Matrix result(rows,cols);
00410   for (int row = 1; row <= rows; row++)
00411     for (int col = 1; col <= cols; col++)
00412       result(row,col) = d * m(row,col);
00413   return result;
00414 
```

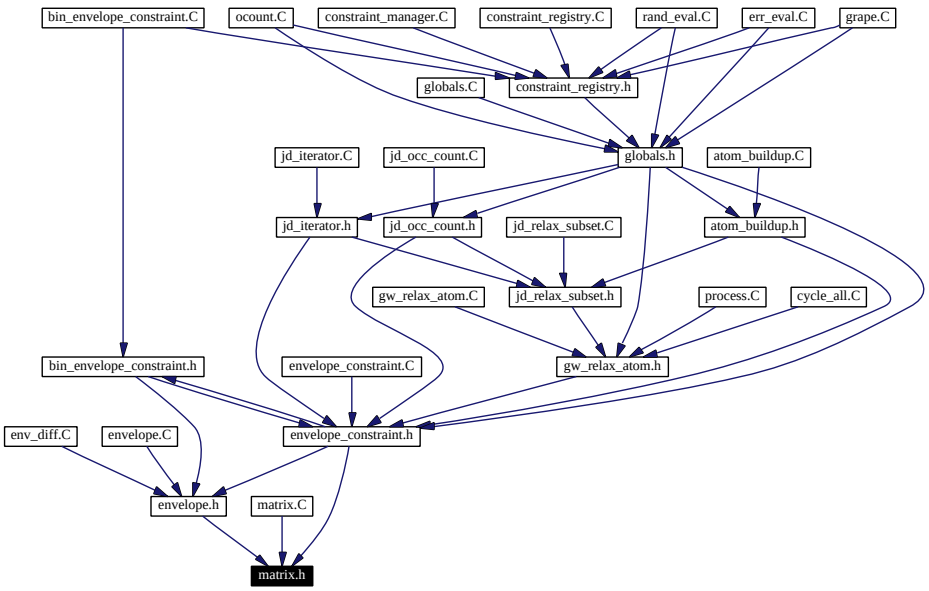
5.87 matrix.h File Reference

```
#include <iostream.h>
#include <iomanip.h>
#include "utils.h"
#include "signal.h"
#include "../error_handler.h"
```

Include dependency graph for matrix.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Matrix**

Functions

- **Matrix operator** ∗ (double d, const **Matrix** &m)
- ostream& **operator**<< (ostream &os, const **Matrix** &m)

5.87.1 Function Documentation

5.87.1.1 Matrix operator * (double *d*, const Matrix & *m*)

Definition at line 407 of file matrix.C.

```
00407                                          {
00408   int rows = m.getRows(), cols = m.getCols();
00409   Matrix result(rows,cols);
00410   for (int row = 1; row <= rows; row++)
00411     for (int col = 1; col <= cols; col++)
00412       result(row,col) = d * m(row,col);
00413   return result;
00414 }
```

5.87.1.2 ostream & operator<< (ostream & *os*, const Matrix & *m*) [inline]

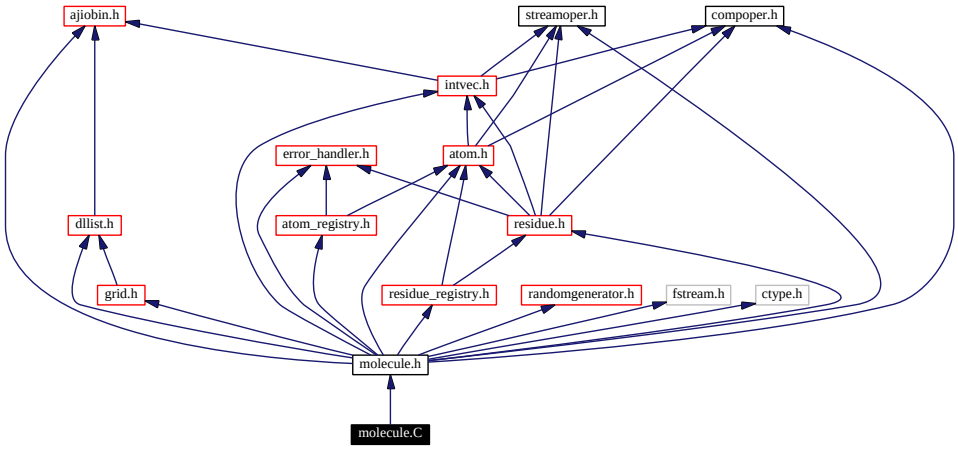
Definition at line 94 of file matrix.h.

```
00094                                                    {
00095   for (int row = 1; row <= m.getRows(); row++) {
00096     os << "[ ";
00097     for (int col = 1; col <= m.getCols(); col++)
00098         os << m(row,col) << " ";
00099     os << "]" << endl;
00100   }
00101 }
```

5.88 molecule.C File Reference

```
#include "molecule.h"
```

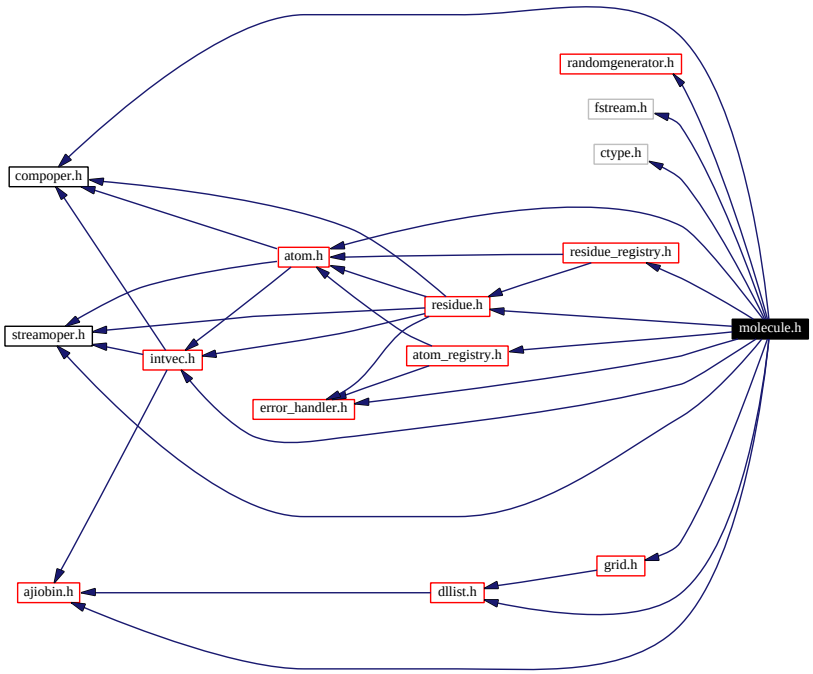
Include dependency graph for molecule.C:



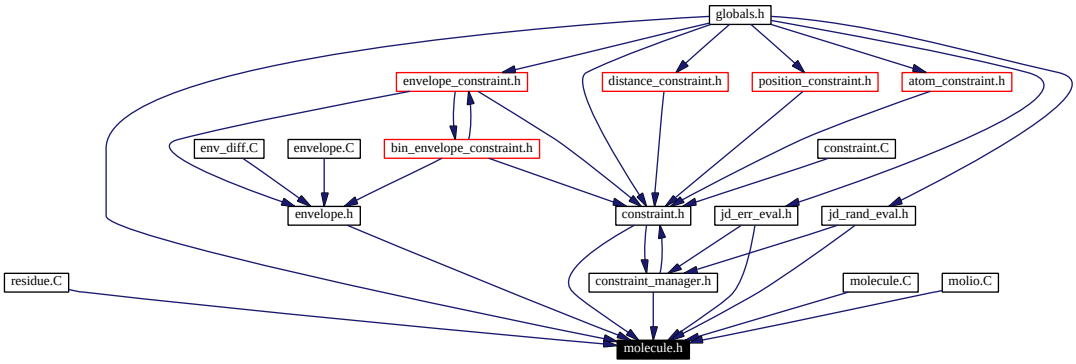
5.89 molecule.h File Reference

```
#include <fstream.h>
#include <ctype.h>
#include "compoper.h"
#include "streamoper.h"
#include "ajiobin.h"
#include "dllist.h"
#include "atom_registry.h"
#include "residue_registry.h"
#include "atom.h"
#include "residue.h"
#include "grid.h"
#include "randomgenerator.h"
#include "error_handler.h"
#include "intvec.h"
```

Include dependency graph for molecule.h:



This graph shows which files directly or indirectly include this file:



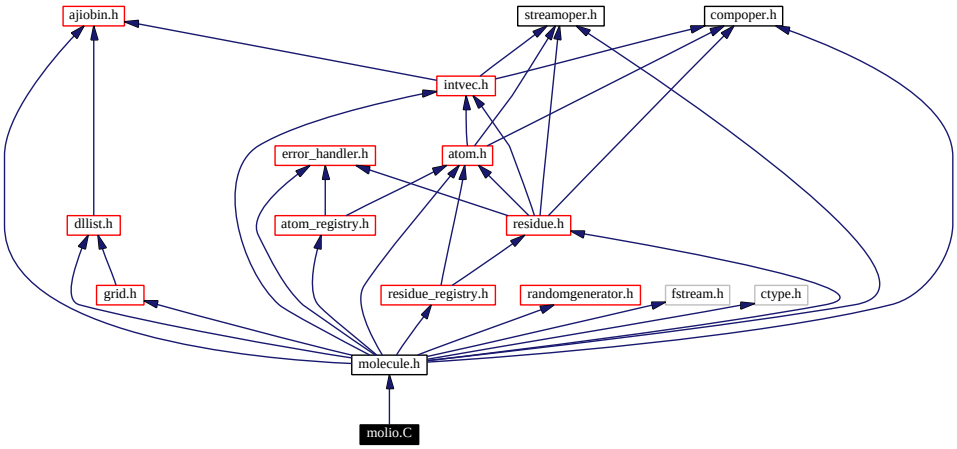
Compounds

- class **Molecule**

5.90 molio.C File Reference

```
#include "molecule.h"
```

Include dependency graph for molio.C:



Functions

- **main** ()

5.90.1 Function Documentation

5.90.1.1 main ()

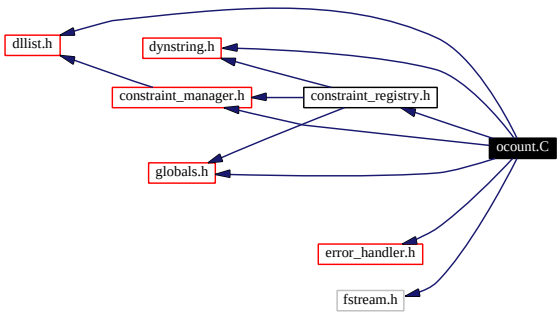
Definition at line 4 of file molio.C.

```
00005   {
00006     Molecule M;
00007     while(!cin.eof() && !cin.bad())
00008       {
00009         M.readXmol(cin);
00010         M.writeXmol(cout);
00011       }
00012
```

5.91 ocount.C File Reference

```
#include <fstream.h>
#include "dynstring.h"
#include "dllist.h"
#include "constraint_manager.h"
#include "constraint_registry.h"
#include "globals.h"
#include "error_handler.h"
```

Include dependency graph for ocount.C:



Functions

- void **print_usage** (ostream &os)

  Prints information about usage.

- void **print_help** (ostream &os)

  Prints full help.

- int **main** (int argc, char **argv)

5.91.1 Function Documentation

5.91.1.1 int main (int *argc*, char ** *argv*)

Definition at line 17 of file ocount.C.

```
00017                                {
00018   if(argc < 2)
00019     {
00020       print_usage(cout);
00021       return 0;
00022     }
00023
00024   cerr << "Begin: ";
00025   for (int i =0; i < argc; i++){
00026     cerr << argv[i]<<" ";
00027   }
```

```
00028   cerr << endl;
00029
00030   DLList<DynString> ArgList;
00031   // List of command line (and parameter file originating) arguments.
00032
00033   Molecule * predictedMolecule=0;
00034   // now the molecule is a private member of CM'
00035
00036   // Iterate through all the command line arguments:
00037   DynString arg;
00038   // Command-line argument.
00039   ifstream * input=0;
00040   // Parameter input stream.
00041   int argStart = 1;
00042
00043   // default output filenamestart string
00044   DynString BaseFN = "ocount-output";
00045
00046
00047   // See if the user has specified a parameter file.  It is a
00048   // parameter file if the first argument does *not* start with
00049   // a dash, '-' (i.e. it is not a flag).
00050   //
00051
00052   if(argv[1][0] != '-')
00053     {
00054       // Read in the arguments from the parameter file.  This file should
00055       // contain flags (including their dashes) followed immediately by
00056       // the values associated with them.  Warning: this processing step
00057       // ignores white space!
00058       //
00059       input = new ifstream(argv[1]);
00060       if(!(input->good()))
00061         {
00062           cerr << "Could not open/find parameter file, " << argv[1]
00063                << ", quitting. Try -help option.\n";
00064           return 1;
00065         }
00066       while(input->good())
00067         {
00068           *input >> arg;
00069           cerr << arg << " ";
00070           if(arg[0] == '#')
00071             arg.EatLine(*input);
00072           // Ignore lines that begin with a pound, '#', sign:
00073           else
00074             ArgList.AddToBack(arg);
00075         }
00076       cerr << endl;
00077       ArgList.RemoveFromBack();  // Get rid of the eof character.
00078       delete input; input=0;
00079       argStart=2;
00080     }
00081
00082   // Copy the arguments from the command line onto our argument list:
00083   //
00084   for(int a=argStart; a<argc; ++a)
00085     ArgList.AddToBack(DynString(argv[a]));
00086
00087   while(ArgList)
00088     {
00089       arg = ArgList.RemoveFromFront();
00090
00091       // Initial molecule file is being specified.
00092       //
```

```
00093
00094       if((arg == DynString("-initial")) ||
00095          (arg == DynString("-init")) ||
00096          (arg == DynString("-i")))
00097         {
00098           // Check to see if we've already read in a molecule:
00099           if(predictedMolecule)
00100             {
00101               cerr << "Warning: More than one file has been supplied for "
00102                    << "the initial molecule.\nWill use the molecule "
00103                    << "specified in the file listed last.\n";
00104               delete predictedMolecule; predictedMolecule=0;
00105             }
00106
00107           arg = ArgList.RemoveFromFront();  // Get the name of the xmol file.
00108           if((input = new ifstream(arg.GetData())))
00109             {
00110               predictedMolecule = new Molecule;
00111               if(predictedMolecule->readXmol(*input)<1)
00112                 {
00113                   cerr << "Error reading initial molecule.\n";
00114                   delete predictedMolecule; predictedMolecule=0;
00115                 }
00116               delete input; input=0;
00117             }
00118           else
00119             {
00120               cerr << "Could not open initial molecule file, " << arg
00121                    << ", will try default location.\n";
00122             }
00123         }
00124
00125       else if(arg == DynString("-f") ||
00126               arg == DynString("-filebase"))
00127         {
00128           arg = ArgList.RemoveFromFront();
00129           if (arg[0] != '-'){
00130             BaseFN = arg;
00131           } else {
00132             DIE("Confused by filename ");
00133           }
00134         }
00135
00136
00137       else if(arg == DynString("-w") ||
00138               arg == DynString("-width"))
00139         {
00140           double width;
00141           if (ArgList.IsEmpty()) {
00142             DIE("Error while parsing the vdw arguments. Try -help option.");
00143           }
00144           arg = ArgList.RemoveFromFront();
00145           width = arg.ToDbl();
00146           jdOcc_Count::cube_width = width;
00147         }
00148
00149       else if(arg == DynString("-help"))
00150         {
00151           print_help(cout);
00152           return 0;
00153
00154         }
00155
00156       // Ignore the bogus parameter.
00157       //
```

```
00158       else
00159         {
00160           cerr << "Unknown argument, " << arg << ", given, ignoring.\n";
00161         }
00162     }
00163
00164   // If we havent read in an initial molecule, we'll check the default
00165   // location for one:
00166   //
00167   if(!predictedMolecule)
00168     {
00169       cerr << "No initial molecule specified, quitting. Try -help option.\n";
00170       return 3;
00171     }
00172
```

5.91.1.2 void print_help (ostream & *os*)

Prints full help.

Definition at line 197 of file ocount.C.

```
00198 {
00199   os    << "\nHELP for using GRAPE\n\n";
00200   print_usage(os);
00201   os    << "Additional Options:\n"
00202         << "  {-w | -width n}
00203
```

5.91.1.3 void print_usage (ostream & *os*)

Prints information about usage.

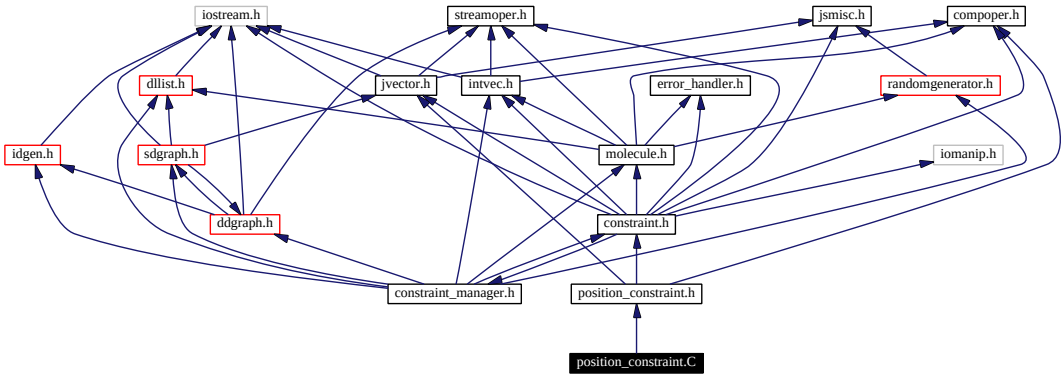
Definition at line 188 of file ocount.C.

```
00189 {
00190   os    << "\nUsage: ocount <-help> [optionfilename | options] \n\n"
00191         << "Required options:\n"
00192         << "  {-initial | -init | -i} <filename> }
00193
```

5.92 position_constraint.C File Reference

```
#include "position_constraint.h"
```

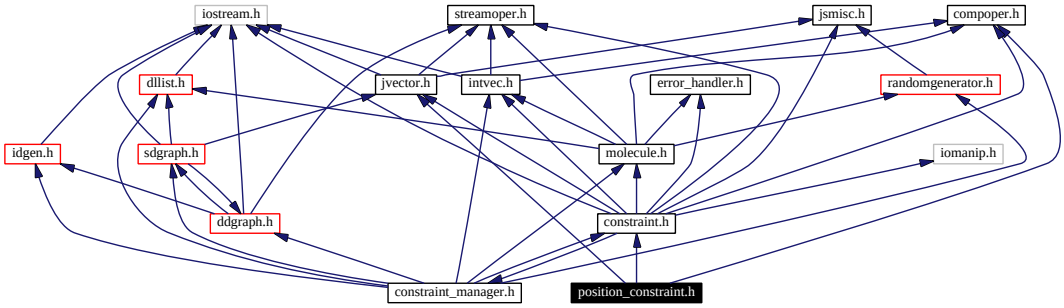
Include dependency graph for position_constraint.C:



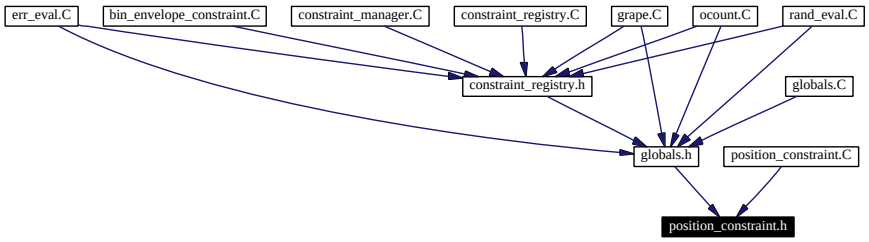
5.93 position_constraint.h File Reference

```
#include "constraint.h"
#include "compoper.h"
#include "jvector.h"
```

Include dependency graph for position_constraint.h:



This graph shows which files directly or indirectly include this file:



Compounds

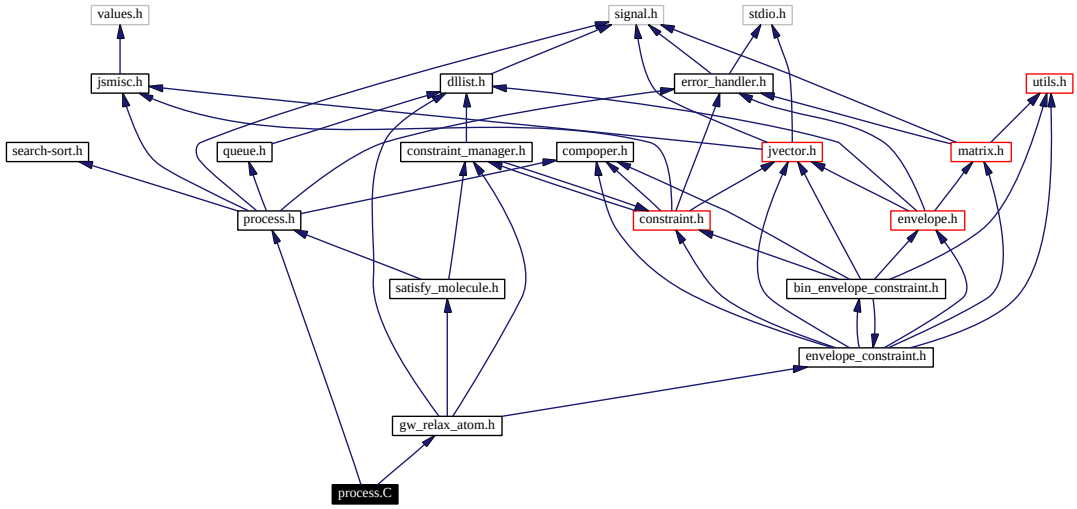
- class **PositionConstraint**

5.94 process.C File Reference

```
#include "process.h"
```

```
#include "gw_relax_atom.h"
```

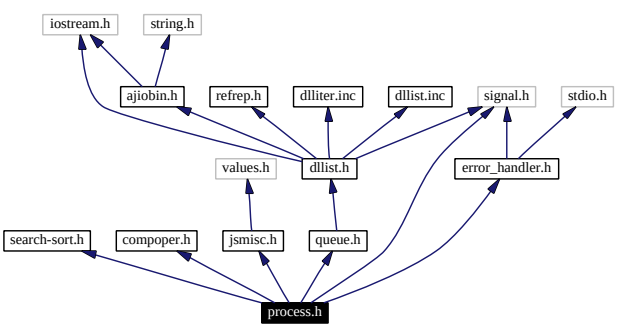
Include dependency graph for process.C:



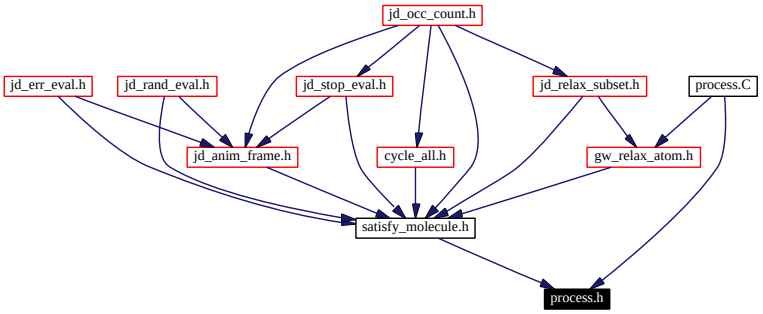
5.95 process.h File Reference

```
#include "search-sort.h"
#include "compoper.h"
#include "jsmisc.h"
#include "queue.h"
#include "signal.h"
#include "error_handler.h"
```

Include dependency graph for process.h:



This graph shows which files directly or indirectly include this file:



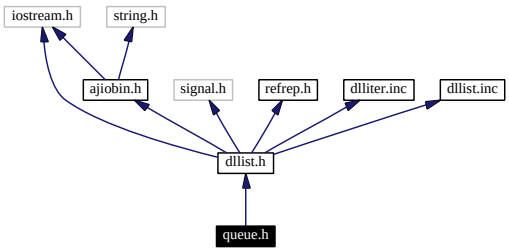
Compounds

- class **Process**

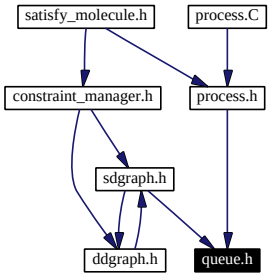
5.96 queue.h File Reference

```
#include "dllist.h"
```

Include dependency graph for queue.h:



This graph shows which files directly or indirectly include this file:



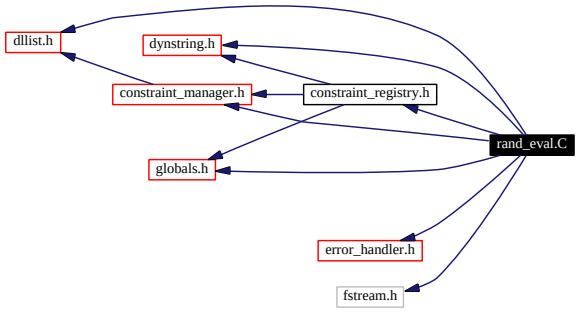
Compounds

- class **Queue**

5.97 rand_eval.C File Reference

```
#include <fstream.h>
#include "dynstring.h"
#include "dllist.h"
#include "constraint_manager.h"
#include "constraint_registry.h"
#include "globals.h"
#include "error_handler.h"
```

Include dependency graph for rand_eval.C:



Functions

- void **print_usage** (ostream &os)
  Prints information about usage.
- void **print_help** (ostream &os)
  Prints full help.
- int **main** (int argc, char **argv)

5.97.1 Function Documentation

5.97.1.1 int main (int *argc*, char ** *argv*)

Definition at line 17 of file rand_eval.C.

```
00017                                {
00018   if(argc < 2)
00019     {
00020       print_usage(cout);
00021       return 0;
00022     }
00023
00024   cerr << "Begin: ";
00025   for (int i =0; i < argc; i++){
00026     cerr << argv[i]<<" ";
00027   }
```

```
00028   cerr << endl;
00029
00031   DLList<DynString> ArgList;
00032
00034   DLList<DynString> ConstraintFileList;
00035
00037   Molecule * predictedMolecule=0;
00038
00040   DynString arg;
00041
00043   ifstream * XmolInputFile=0;
00044
00045
00047   ifstream * input=0;
00048
00050   int argStart = 1;
00051
00053   DynString BaseFN = "evalenv-output";
00054
00055   int cycle = 10;
00056   double pert = 0.05;
00057
00058   // See if the user has specified a parameter file.  It is a
00059   // parameter file if the first argument does *not* start with
00060   // a dash, '-' (i.e. it is not a flag).
00061   //
00062
00063   if(argv[1][0] != '-')
00064     {
00065       // Read in the arguments from the parameter file.  This file should
00066       // contain flags (including their dashes) followed immediately by
00067       // the values associated with them.  Warning: this processing step
00068       // ignores white space!
00069       //
00070       if(!(input = new ifstream(argv[1])))
00071         {
00072           cerr << "Could not open/find parameter file, " << argv[1]
00073                << ", quitting. Try -help option.\n";
00074           return 1;
00075         }
00076       while(!input->eof() && !input->bad())
00077         {
00078           *input >> arg;
00079
00080           if(arg[0] == '#')
00081             arg.EatLine(*input);
00082           // Ignore lines that begin with a pound, '#', sign:
00083           else
00084             ArgList.AddToBack(arg);
00085         }
00086       ArgList.RemoveFromBack();  // Get rid of the eof character.
00087
00088       delete input; input=0;
00089       argStart=2;
00090     }
00091
00092   // Copy the arguments from the command line onto our argument list:
00093   //
00094   for(int a=argStart; a<argc; ++a)
00095     ArgList.AddToBack(DynString(argv[a]));
00096
00097   while(ArgList)
00098     {
00099       arg = ArgList.RemoveFromFront();
00100
```

```
00101       // Initial molecule file is being specified.
00102       //
00103
00104       if((arg == DynString("-initial")) ||
00105          (arg == DynString("-init")) ||
00106          (arg == DynString("-i")))
00107         {
00108           // Check to see if we've already read in a molecule:
00109           if(predictedMolecule)
00110             {
00111               cerr << "Warning: More than one file has been supplied for "
00112                    << "the initial molecule.\nWill use the molecule "
00113                    << "specified in the file listed last.\n";
00114               delete predictedMolecule; predictedMolecule=0;
00115             }
00116
00117           arg = ArgList.RemoveFromFront();  // Get the name of the xmol file.
00118           if((XmolInputFile = new ifstream(arg.GetData())))
00119             {
00120               predictedMolecule = new Molecule;
00121               if(predictedMolecule->readXmol(*XmolInputFile)<1)
00122                 {
00123                   cerr << "Error reading initial molecule.\n";
00124                   delete predictedMolecule; predictedMolecule=0;
00125                 }
00126             }
00127           else
00128             {
00129               cerr << "Could not open initial molecule file, " << arg
00130                    << ", will try default location.\n";
00131             }
00132         }
00133
00134       else if((arg == DynString("-constraint")) ||
00135               (arg == DynString("-constraints")) ||
00136               (arg == DynString("-c")))
00137         {
00138           // Just add this file to the list of files that contain
00139           // constraints for right now.  We'll read these in once
00140           // we're sure we have an initial molecule.
00141           //
00142           arg = ArgList.RemoveFromFront();
00143           ConstraintFileList.AddToBack(arg);
00144         }
00145
00146       else if(arg == DynString("-f") ||
00147               arg == DynString("-filebase"))
00148         {
00149             arg = ArgList.RemoveFromFront();
00150             if (arg[0] != '-'){
00151               BaseFN = arg;
00152             } else {
00153               DIE("Confused by filename ");
00154             }
00155         }
00156
00157       else if(arg == DynString("-cycle"))
00158         {
00159           arg = ArgList.RemoveFromFront();
00160           if (arg[0] != '-'){
00161             cycle = arg.ToInt();
00162           } else {
00163             DIE("Confused by cycle arg ");
00164           }
00165         }
```

```
00166
00167       else if(arg == DynString("-pert"))
00168         {
00169           arg = ArgList.RemoveFromFront();
00170           if (arg[0] != '-'){
00171             pert = arg.ToDbl();
00172           } else {
00173             DIE("Confused by pert arg ");
00174           }
00175         }
00176
00177       else if(arg == DynString("-help"))
00178         {
00179           print_help(cout);
00180           return 0;
00181
00182         }
00183
00184       // Ignore the bogus parameter.
00185       //
00186       else
00187         {
00188           cerr << "Unknown argument, " << arg << ", given, ignoring.\n";
00189         }
00190     }
00191
00192   // If we havent read in an initial molecule, we'll check the default
00193   // location for one:
00194   //
00195   if(!predictedMolecule)
00196     {
00197       cerr << "No initial molecule specified, quitting. Try -help option.\n";
00198       return 3;
00199     }
00200
00201   // Now that we've got an actual molecule, we can create a constraint
00202   // manager and read in the constraints:
00203   //
00204
00205   ConstraintManager CM(*predictedMolecule);
00206
00207   int num_read = 0;
00208
00209   while(ConstraintFileList)
00210     {
00211       // Get the filename and then open the file and then read the
00212       // constraints in using the constraint manager.
00213       arg = ConstraintFileList.RemoveFromFront();
00214       input = new ifstream(arg.GetData());
00215       if(input->good())
00216         {
00217           // Use the registry's routine to read in constraints and
00218           // append them to the constraint manager.
00219           //
00220
00221           num_read += ConstraintRegistry::readConstraints(*input,CM);
00222           delete input; input=0;
00223         } else {
00224           cerr << "\n\nWhile reading file: " << arg.GetData() << endl;
00225           DIE ("Cannot open file");
00226         }
00227
00228     }
00229
00230   if (num_read == 0) {
```

```
00231     DIE ("You have no constraints");
00232   }
00233
00234   jdRand_Eval myeval(CM, XmolInputFile, cycle, pert);
00235   myeval.BaseFN = BaseFN;
00236   myeval.run();
00237
00238   delete predictedMolecule;
00239   if (XmolInputFile)
00240     {delete XmolInputFile;XmolInputFile =0;}
00241
```

5.97.1.2 void print_help (ostream & *os*)

Prints full help.

Definition at line 260 of file rand_eval.C.

Referenced by main().

```
00261 {
00262   os    << "\nHELP\n\n";
00263   print_usage(os);
00264   os    << "Additional Options:\n"
00265         << "  -cycle n   # perturb for n cycles\n"
00266         << "  -pert  n    # move each atom up to +/- n Angstoms in each direction\n";
00267
00268
```

5.97.1.3 void print_usage (ostream & *os*)

Prints information about usage.

Definition at line 246 of file rand_eval.C.

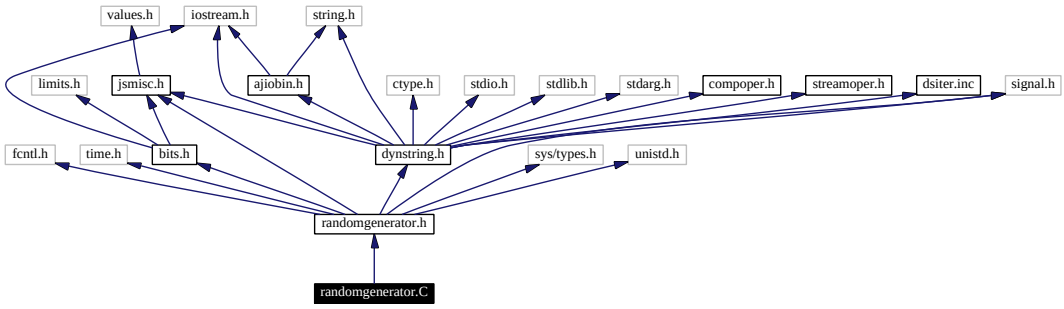
Referenced by main(), and print_help().

```
00247 {
00248   os    << "\nUsage: rand_eval <-help> [optionfilename | options] \n\n"
00249         << "Required options:\n"
00250         << "  {-initial | -init | -i} <filename> }\n"
00251         << "     Initial molecule: xmol file\n"
00252         << "  {-constraints | -c}
00253
```

5.98 randomgenerator.C File Reference

```
#include "randomgenerator.h"
```

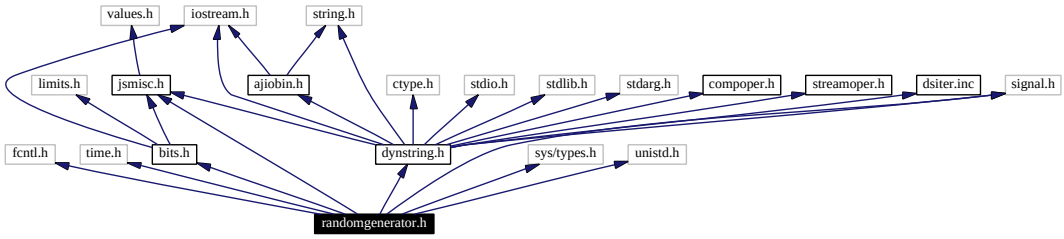
Include dependency graph for randomgenerator.C:



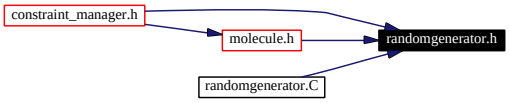
5.99 randomgenerator.h File Reference

```
#include <sys/types.h>
#include <unistd.h>
#include <fcntl.h>
#include <time.h>
#include <bits.h>
#include <signal.h>
#include "jsmisc.h"
#include "dynstring.h"
```

Include dependency graph for randomgenerator.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **RanBaseClass**
- class **RanChar**
- class **RanDbl**
- class **RanInt**
- class **RanLetter**
- class **RanWord**
- class **z**

Variables

- class z **z**

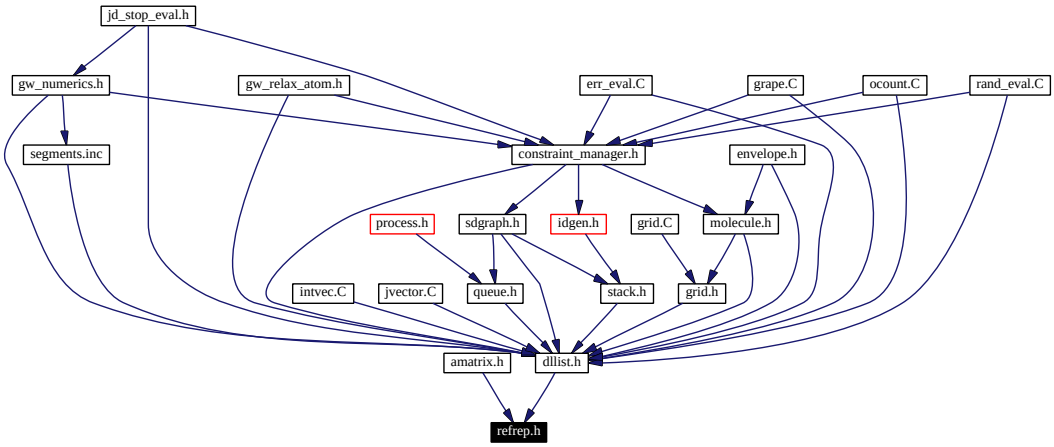
5.99.1 Variable Documentation

5.99.1.1 class z z [static]

Referenced by Matrix::findEigenVectors().

5.100 refrep.h File Reference

This graph shows which files directly or indirectly include this file:



Compounds

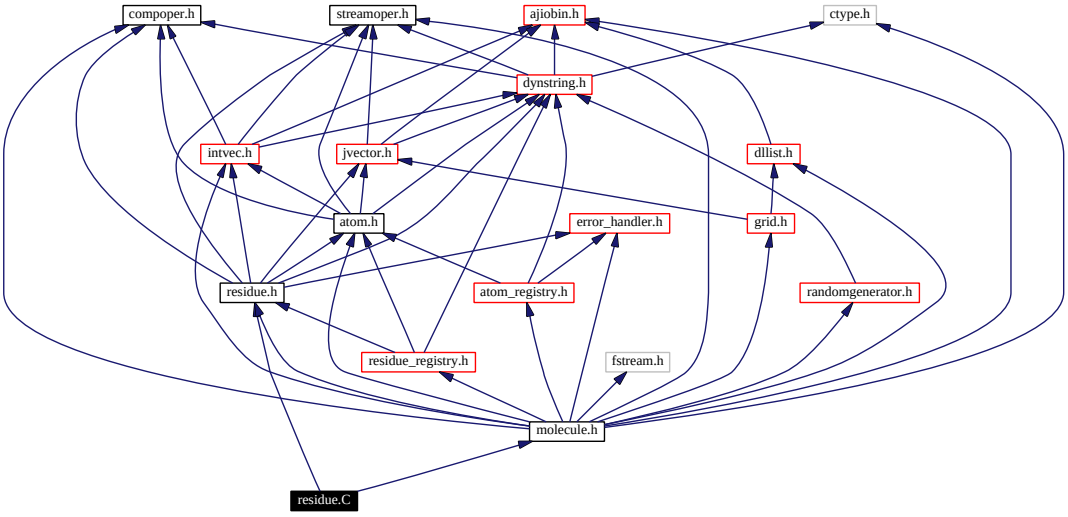
- class **RefRep**

5.101 residue.C File Reference

```
#include "residue.h"
```

```
#include "molecule.h"
```

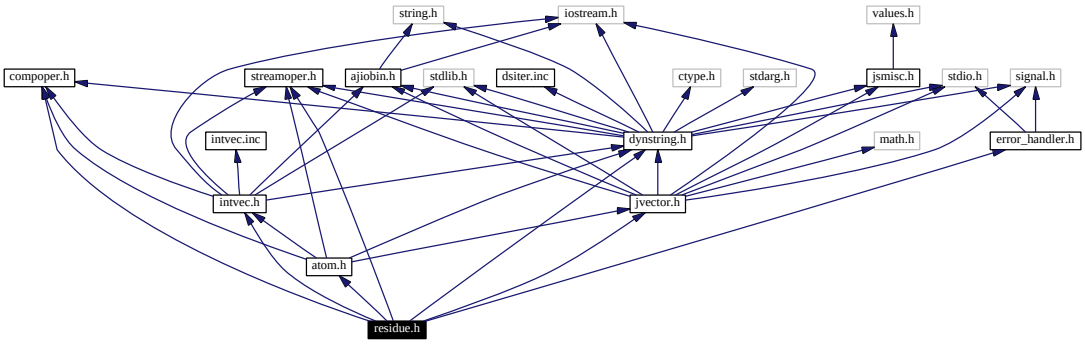
Include dependency graph for residue.C:



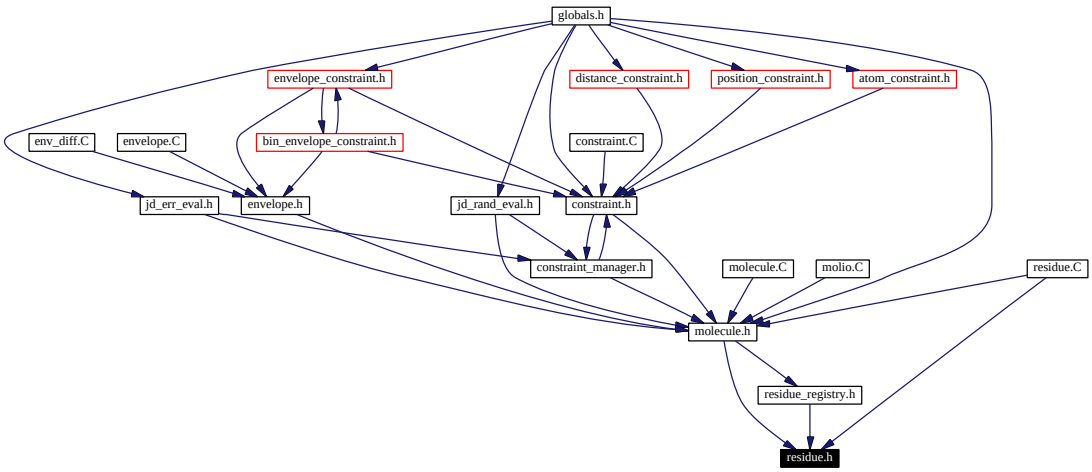
5.102 residue.h File Reference

```
#include "intvec.h"
#include "dynstring.h"
#include "atom.h"
#include "jvector.h"
#include "compoper.h"
#include "streamoper.h"
#include "error_handler.h"
```

Include dependency graph for residue.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **Residue**

Defines

- #define **UNINITIALIZED_RES_ID** -1

5.102.1 Define Documentation

5.102.1.1 #define UNINITIALIZED_RES_ID -1

//

File: **residue.h**

Used to save information about one residue.

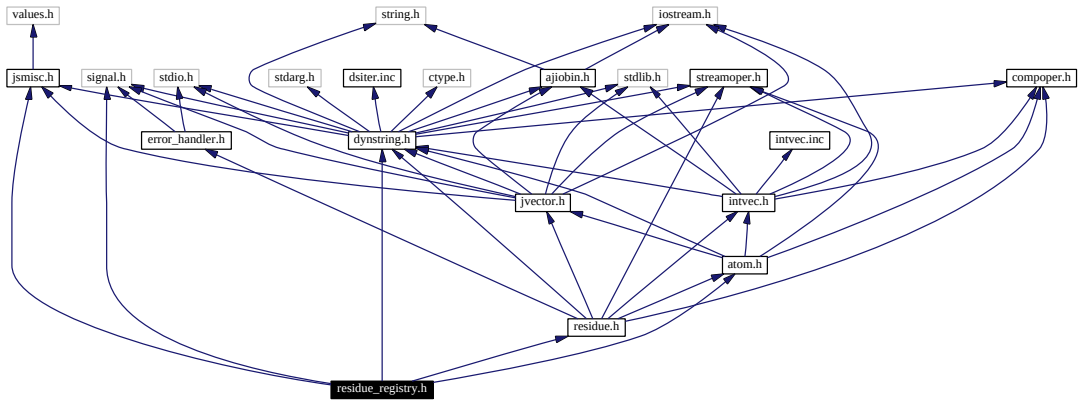
//

Definition at line 23 of file residue.h.

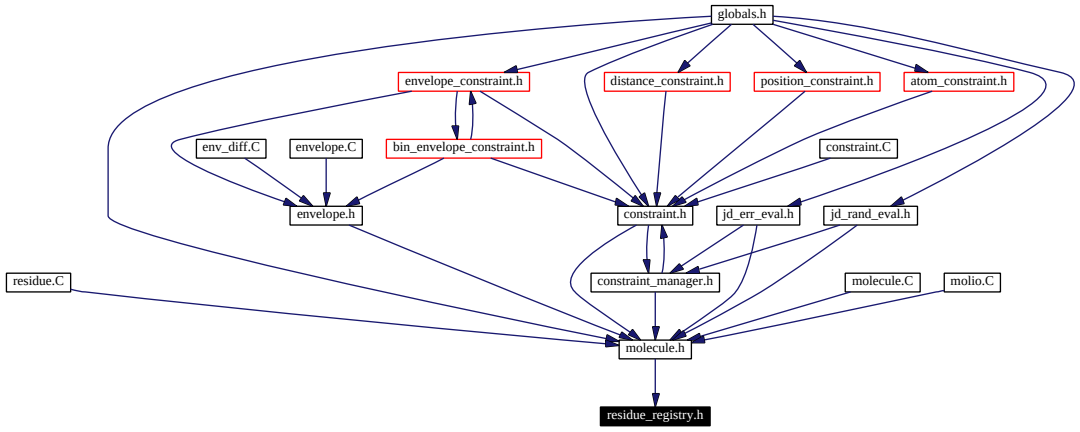
5.103 residue_registry.h File Reference

```
#include <signal.h>
#include "dynstring.h"
#include "residue.h"
#include "atom.h"
#include "jsmisc.h"
```

Include dependency graph for residue_registry.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **ResidueRegistry**

Defines

- #define **num_known_residue_species** 21

5.103.1 Define Documentation

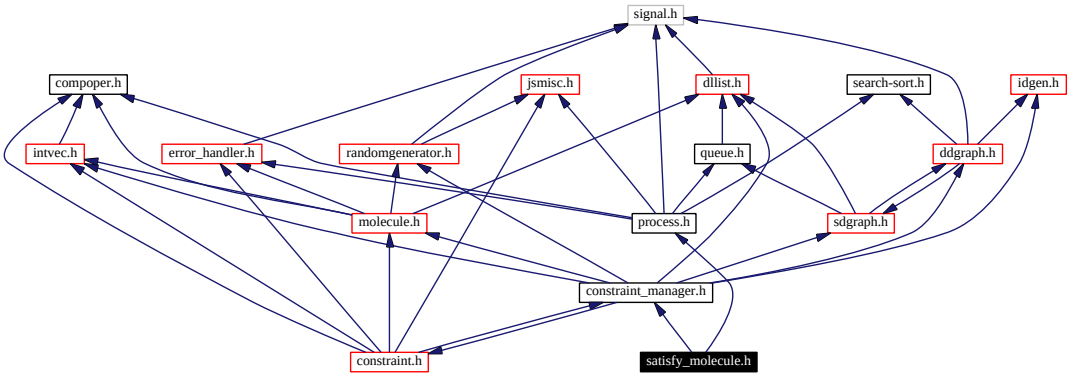
5.103.1.1 #define num_known_residue_species 21

Definition at line 17 of file residue_registry.h.

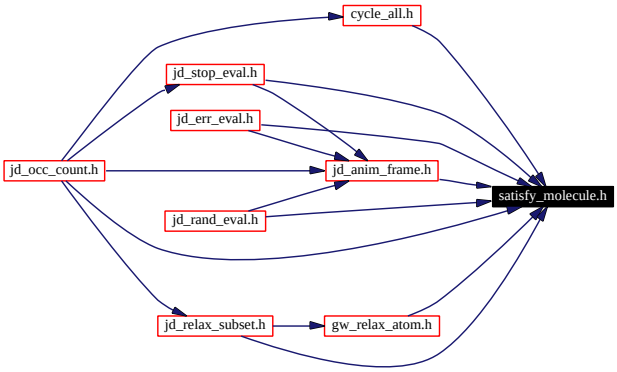
5.104 satisfy_molecule.h File Reference

```
#include "process.h"
#include "constraint_manager.h"
```

Include dependency graph for satisfy_molecule.h:



This graph shows which files directly or indirectly include this file:



Compounds

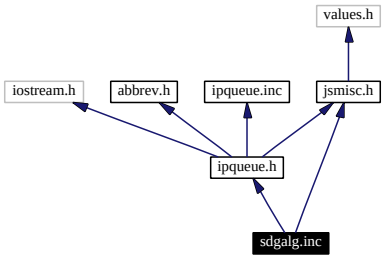
- class **SatisfyMolecule**

5.105 sdgalg.inc File Reference

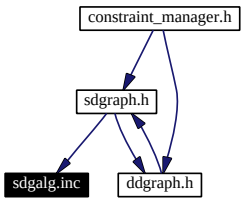
```
#include "ipqueue.h"
```

```
#include "jsmisc.h"
```

Include dependency graph for sdgalg.inc:

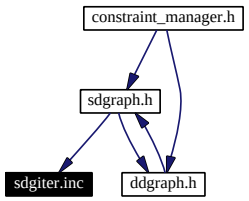


This graph shows which files directly or indirectly include this file:



5.106 sdgiter.inc File Reference

This graph shows which files directly or indirectly include this file:



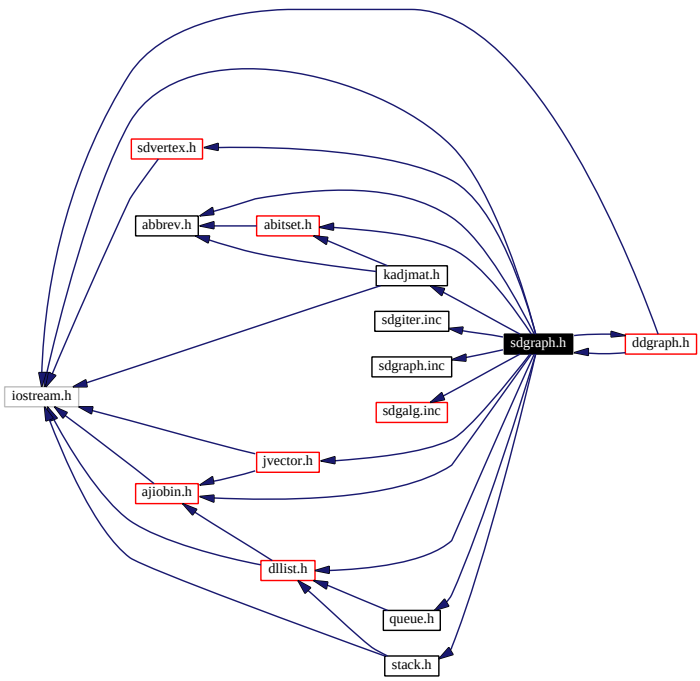
Compounds

- class **base_iterator**
- class **bf_iterator**
- class **df_iterator**
- class **neighbor_iterator**
- class **path_iterator**

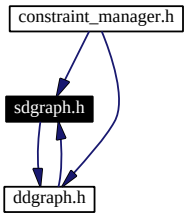
5.107 sdgraph.h File Reference

```
#include <iostream.h>
#include "sdvertex.h"
#include "ddgraph.h"
#include "dllist.h"
#include "stack.h"
#include "queue.h"
#include "kadjmat.h"
#include "abbrev.h"
#include "abitset.h"
#include "ajiobin.h"
#include "jvector.h"
#include <sdgiter.inc>
#include <sdgraph.inc>
#include <sdgalg.inc>
```

Include dependency graph for sdgraph.h:



This graph shows which files directly or indirectly include this file:



Compounds

- class **sdGraph**

Typedefs

- typedef set<int> **SDG_SetInt**
- typedef set<**SDG_SetInt**> **SDG_SetSetInt**
- typedef **DLList**<int> **SDG_IntList**
- typedef **sdGraph**<**SDG_SetInt**, class WT> **SDG_GraphSetInt**

Enumerations

- enum **sdg_color** { **sdg_white** = 0, **sdg_gray** = 1, **sdg_black** = 2 }

5.107.1 Typedef Documentation

5.107.1.1 typedef sdGraph< SDG_SetInt, class WT > SDG_GraphSetInt

Definition at line 63 of file sdgraph.h.

5.107.1.2 typedef DLList< int > SDG_IntList

Definition at line 41 of file sdgraph.h.

5.107.1.3 typedef set< int > SDG_SetInt

Definition at line 38 of file sdgraph.h.

5.107.1.4 typedef set< SDG_SetInt > SDG_SetSetInt

Definition at line 40 of file sdgraph.h.

5.107.2 Enumeration Type Documentation

5.107.2.1 enum sdg_color

Enumeration values:

sdg_white

sdg_gray

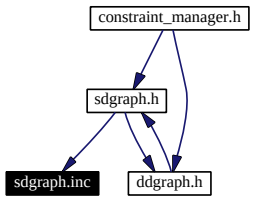
sdg_black

Definition at line 65 of file sdgraph.h.

```
00066 {
00067   sdg_white     = 0,
00068   sdg_gray      = 1,
00069   sdg_black     = 2
00070
```

5.108 sdgraph.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class DT, class WT> ostream& **operator**<< (ostream &os, const **sdGraph**< DT, WT > &graph)
- istream& **operator**>= (istream &is, **sdGraph**< DT, WT > &sdg)
- template<class DT, class WT> ostream& **operator**<= (ostream &os, const **sdGraph**< DT, WT > &sdg)

5.108.1 Function Documentation

5.108.1.1 template<class DT, class WT> ostream & operator<< (ostream & *os*, const sdGraph< DT, WT > & *graph*) [inline]

Definition at line 661 of file sdgraph.inc.

```
00662   { graph.Output(os); return os; }
00663
```

5.108.1.2 template<class DT, class WT> ostream & operator<= (ostream & *os*, const sdGraph< DT, WT > & *sdg*) [inline]

Definition at line 948 of file sdgraph.inc.

```
00949   { sdg.OutputBinary(os); return os; }
00950
```

5.108.1.3 istream & operator>= (istream & *is*, sdGraph< DT, WT > & *sdg*) [inline]

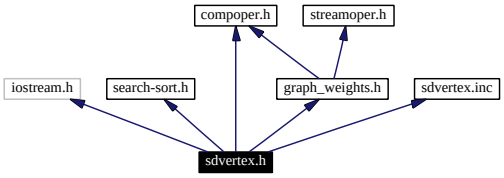
Definition at line 944 of file sdgraph.inc.

```
00945   { sdg.InputBinary(is); return is; }
00946
```

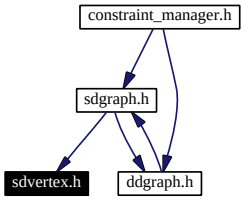
5.109 sdvertex.h File Reference

```
#include <iostream.h>
#include <search-sort.h>
#include <compoper.h>
#include <graph_weights.h>
#include <sdvertex.inc>
```

Include dependency graph for sdvertex.h:



This graph shows which files directly or indirectly include this file:

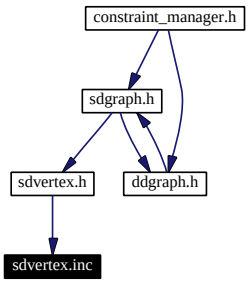


Compounds

- class **sdEdge**
- class **sdVertex**

5.110 sdvertex.inc File Reference

This graph shows which files directly or indirectly include this file:



Functions

- template<class WT> ostream& **operator**<< (ostream &os, const **sdVertex**< WT > &vert)
- int **Connected** (const **sdVertex**< WT > &v1, const **sdVertex**< WT > &v2)

5.110.1 Function Documentation

5.110.1.1 int Connected (const sdVertex< WT > & *v1*, const sdVertex< WT > & *v2*) [inline]

Definition at line 244 of file sdvertex.inc.

```
00245   { return (v1.HasEdgeTo(v2) || v1.HasEdgeFrom(v2)); }
00246
```

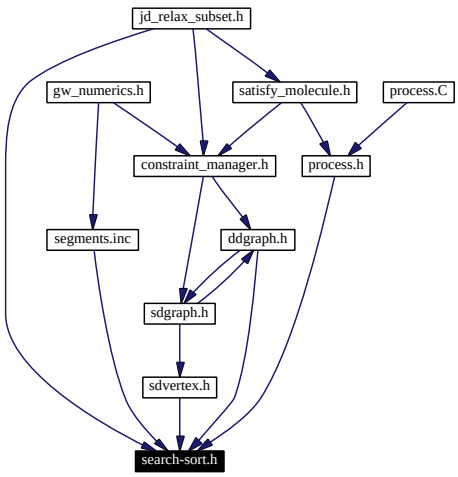
5.110.1.2 template<class WT> ostream & operator<< (ostream & *os*, const sdVertex< WT > & *vert*) [inline]

Definition at line 122 of file sdvertex.inc.

```
00123   { os << vert.ID; return os; }
00124
```

5.111 search-sort.h File Reference

This graph shows which files directly or indirectly include this file:



Functions

- void **search_swap** (T &t1, T &t2)
- void **BubbleSort** (T *A, int num)
- void **BubbleSortPtrs** (T **A, int num)
- void **BubbleSortR** (T *A, int num)
- void **BubbleSortPtrsR** (T **A, int num)
- int **Partition** (T *A, int p, int r)
- int **PartitionPtrs** (T **A, int p, int r)
- void **RecQuickSort** (T *A, int p, int r)
- void **RecQuickSortPtrs** (T **A, int p, int r)
- void **QuickSort** (T *A, int Num)
- void **QuickSortPtrs** (T **A, int Num)
- int **BinSearch** (const T *A, int Num, const T &x)
- int **BinSearchPtr** (T **A, int Num, const T &x)
- int **SelectPartition** (T *A, int p, int r, T x)
- double **LinearSelect** (double *L, int k, int n)

5.111.1 Function Documentation

5.111.1.1 int BinSearch (const T * *A*, int *Num*, const T & *x*) [inline]

Definition at line 192 of file search-sort.h.

Referenced by ddGraph::Copy(), sdVertex::GetInEdgeIndex(), sdVertex::GetOutEdgeIndex(), and sdGraph::GetVertexIndex().

```
00193   {
00194     // left index, right index, pivot, respectively.
00195     //
00196     int i=0;
00197     int j=Num-1;
```

```
00198     int p;
00199
00200     while((i<=j) && (j>=0))
00201       {
00202         p = (j+i)/2;
00203         if(x < A[p]) j = p-1;
00204         else if(x > A[p]) i = p+1;
00205         else return p;
00206       }
00207
```

5.111.1.2 int BinSearchPtr (T ** *A*, int *Num*, const T & *x*) [inline]

Definition at line 211 of file search-sort.h.

```
00212   {
00213     // left index, right index, pivot, respectively.
00214     //
00215     int i=0;
00216     int j=Num-1;
00217     int p;
00218
00219     while((i<=j) && (j>=0))
00220       {
00221         p = (j+i)/2;
00222         if(x < *(A[p])) j = p-1;
00223         else if(x > *(A[p])) i = p+1;
00224         else return p;
00225       }
00226
```

5.111.1.3 void BubbleSort (T * *A*, int *num*) [inline]

Definition at line 51 of file search-sort.h.

```
00051                                     {
00052   bool stillGoing;
00053   do {
00054     stillGoing = false;
00055     for( int i=0; i<num-1; i++ ) {
00056       if( A[i] > A[i+1] ) {
00057         search_swap( A[i], A[i+1] );
00058         stillGoing = true;
00059       }
00060     }
00061   }
00062
```

5.111.1.4 void BubbleSortPtrs (T ** *A*, int *num*) [inline]

Definition at line 65 of file search-sort.h.

```
00065                                          {
00066   bool stillGoing;
00067   do {
00068     stillGoing = false;
00069     for( int i=0; i<num-1; i++ ) {
00070       if( *(A[i]) > *(A[i+1]) ) {
```

```
00071         search_swap( A[i], A[i+1] );
00072         stillGoing = true;
00073       }
00074     }
00075   }
00076
```

5.111.1.5 void BubbleSortPtrsR (T ** *A*, int *num*) [inline]

Definition at line 93 of file search-sort.h.

```
00093                                          {
00094   bool stillGoing;
00095   do {
00096     stillGoing = false;
00097     for( int i=num-1;i>0; i-- ) {
00098       if( *(A[i]) < *(A[i-1]) ) {
00099         search_swap( A[i], A[i-1] );
00100         stillGoing = true;
00101       }
00102     }
00103   }
00104
```

5.111.1.6 void BubbleSortR (T * *A*, int *num*) [inline]

Definition at line 79 of file search-sort.h.

```
00079                                       {
00080   bool stillGoing;
00081   do {
00082     stillGoing = false;
00083     for( int i=num-1; i>0; i-- ) {
00084       if( A[i] < A[i-1] ) {
00085         search_swap( A[i], A[i-1] );
00086         stillGoing = true;
00087       }
00088     }
00089   }
00090
```

5.111.1.7 double LinearSelect (double * *L*, int *k*, int *n*) [inline]

Definition at line 263 of file search-sort.h.

```
00263                                                  {
00264   // returns the k'th smallest element in L.
00265   // !! k is in the set {1 -> n} for an n element list. !! (not 0->n-1)
00266   // n represents the number of elements in L.
00267
00268   int i;
00269   if (n <=10) {
00270     RecQuickSort(L, 0, n-1);
00271     return L[k-1];
00272   }
00273   int N = (int) ((n+4)/5);
00274   double ** S = new double * [N];
00275   int j = 0;
```

```
00276   for (i = 0; i < (N-1); i++){
00277     S[i] = new double [5];
00278     for (int u =0; u<5; u++) S[i][u] = L[j++];
00279   }
00280   S[N-1] = new double [5];
00281   int rem = n-j;
00282   for (i = 0; i < rem; i++){
00283     S[N-1][i] = L[j+i];
00284   }
00285   double * X = new double [N];
00286   for (i = 0;i<N-1;i++){
00287     RecQuickSort(S[i], 0, 4);
00288     X[i] = S[i][2];
00289   }
00290   if (rem == 1) X[N-1] = S[i][0];
00291   else {
00292     RecQuickSort(S[i], 0, rem-1);
00293     if (rem == 4) X[N-1] = S[i][2];
00294     else X[N-1] = S[i][1];
00295   }
00296   for (i = 0;i<N;i++){
00297     delete [] S[i];
00298   }
00299
```

5.111.1.8 int Partition (T * *A*, int *p*, int *r*) [inline]

Definition at line 107 of file search-sort.h.
Referenced by RecQuickSort().

```
00108   {
00109     T tmp;
00110     T x = A[p];
00111     int i = p-1;
00112     int j = r+1;
00113
00114     while(1)
00115       {
00116         --j;
00117         while(A[j]>x) { --j; }
00118         ++i;
00119         while(A[i]<x) { ++i; }
00120
00121         //
00122         // Swap the values at i and j.
00123         if(i<j)
00124           {
00125             tmp  = A[i];
00126             A[i] = A[j];
00127             A[j] = tmp;
00128           }
00129         else return j;
00130       }
00131
```

5.111.1.9 int PartitionPtrs (T ** *A*, int *p*, int *r*) [inline]

Definition at line 134 of file search-sort.h.
Referenced by RecQuickSortPtrs().

```
00135   {
00136     T * tmp;
00137     T * x = A[p];
00138     int i = p-1;
00139     int j = r+1;
00140
00141     while(1)
00142       {
00143         --j;
00144         while(*(A[j])>*x) { --j; }
00145         ++i;
00146         while(*(A[i])<*x) { ++i; }
00147
00148         if(i<j)
00149           {
00150             tmp  = A[i];
00151             A[i] = A[j];
00152             A[j] = tmp;
00153           }
00154         else return j;
00155       }
00156
```

5.111.1.10 void QuickSort (T * *A*, int *Num*) [inline]

Definition at line 181 of file search-sort.h.

Referenced by sdGraph::Copy(), sdVertex::SortEdges(), and sdGraph::SubGraph().

```
00182   {
00183     if(Num<2) return;
00184     RecQuickSort(A, 0, Num-1);
00185
```

5.111.1.11 void QuickSortPtrs (T ** *A*, int *Num*) [inline]

Definition at line 188 of file search-sort.h.

Referenced by Process::assertOrder().

```
00189   { if(Num<2) return; RecQuickSortPtrs(A,0,Num-1); }
00190
```

5.111.1.12 void RecQuickSort (T * *A*, int *p*, int *r*) [inline]

Definition at line 159 of file search-sort.h.

Referenced by LinearSelect(), Segments::QSortReduce(), and QuickSort().

```
00160   {
00161     if(p<r)
00162       {
00163         int q = Partition(A,p,r);
00164         RecQuickSort(A,p,q);
00165         RecQuickSort(A,q+1,r);
00166       }
00167
```

5.111.1.13 void RecQuickSortPtrs (T ** *A*, int *p*, int *r*) [inline]

Definition at line 170 of file search-sort.h.
Referenced by QuickSortPtrs().

```
00171   {
00172     if(p<r)
00173       {
00174         int q = PartitionPtrs(A,p,r);
00175         RecQuickSortPtrs(A,p,q);
00176         RecQuickSortPtrs(A,q+1,r);
00177       }
00178
```

5.111.1.14 int SelectPartition (T * *A*, int *p*, int *r*, T *x*) [inline]

Definition at line 230 of file search-sort.h.
Referenced by LinearSelect().

```
00231   {
00232     // as above, but able to specify partion element
00233     T tmp;
00234     int i = p-1;
00235     int j = r+1;
00236
00237     while(1)
00238       {
00239         --j;
00240         while(A[j]>x) { --j; }
00241         ++i;
00242         while(A[i]<x) { ++i; }
00243
00244         //
00245         // Swap the values at i and j.
00246         if(i<j)
00247           {
00248             tmp  = A[i];
00249             A[i] = A[j];
00250             A[j] = tmp;
00251           }
00252         else return j;
00253       }
00254
```

5.111.1.15 void search_swap (T & *t1*, T & *t2*) [static]

Definition at line 42 of file search-sort.h.
Referenced by BubbleSort(), BubbleSortPtrs(), BubbleSortPtrsR(), and BubbleSortR().

```
00043 {
00044         T t = t1;
00045         t1 = t2;
00046         t2 = t;
00047
```

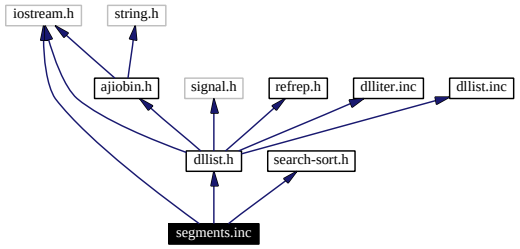
5.112 segments.inc File Reference

```
#include <iostream.h>
```

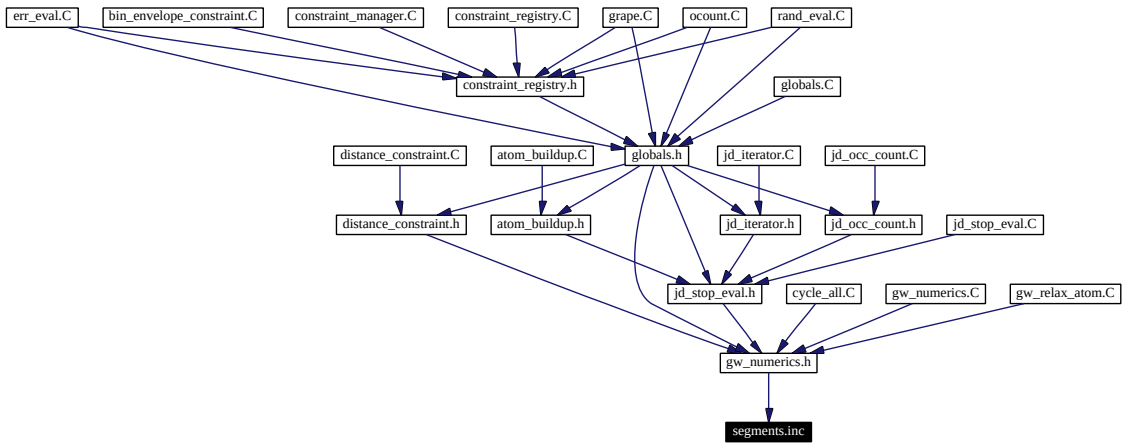
```
#include "dllist.h"
```

```
#include "search-sort.h"
```

Include dependency graph for segments.inc:



This graph shows which files directly or indirectly include this file:



Compounds

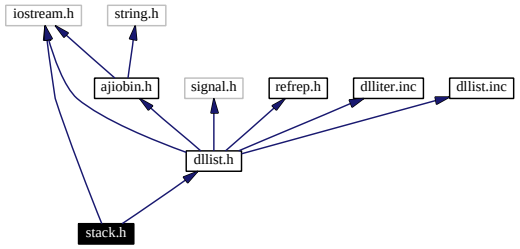
- class **Segment**
- class **Segments**

5.113 stack.h File Reference

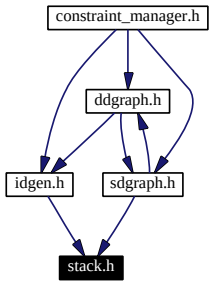
```
#include <iostream.h>
```

```
#include "dllist.h"
```

Include dependency graph for stack.h:



This graph shows which files directly or indirectly include this file:

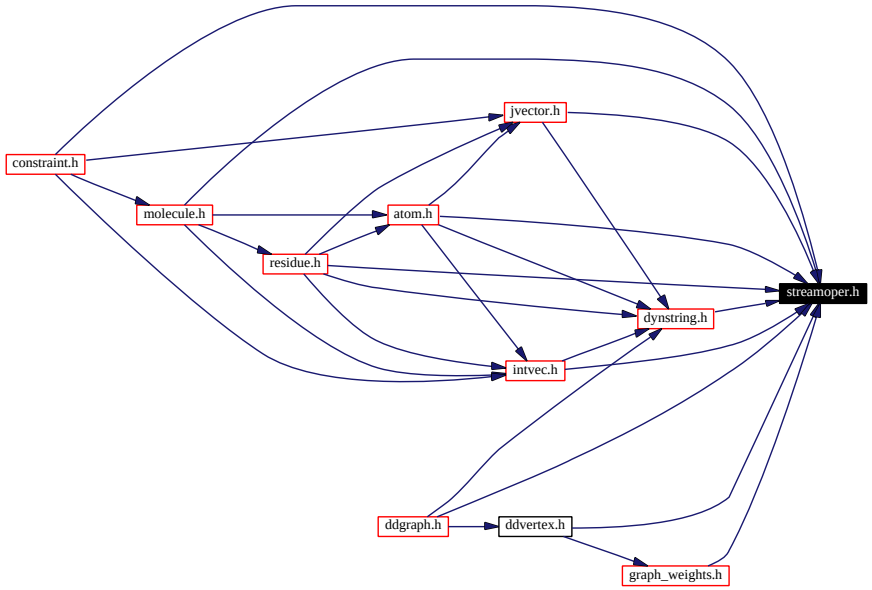


Compounds

- class **Stack**

5.114 streamoper.h File Reference

This graph shows which files directly or indirectly include this file:



Defines

- #define **MAKE_IO_STREAM_OPERATORS**(CLASSTYPE_XX)
- #define **MAKE_BIN_STREAM_OPERATORS**(CLASSTYPE_XX)

5.114.1 Define Documentation

5.114.1.1 #define MAKE_BIN_STREAM_OPERATORS(CLASSTYPE_XX)

Value:

```
friend ostream& operator <= (ostream& os, const CLASSTYPE_XX& x) \
        { x.OutputBinary(os); return os; } \
friend istream& operator >= (istream& is, CLASSTYPE_XX& x) \
        { x.InputBinary(is); return is; }
```

Definition at line 16 of file streamoper.h.

5.114.1.2 #define MAKE_IO_STREAM_OPERATORS(CLASSTYPE_XX)

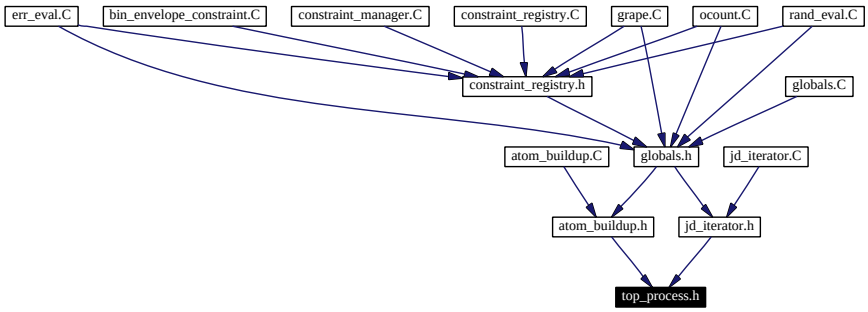
Value:

```
friend ostream& operator << (ostream& os, const CLASSTYPE_XX& x) \
        { x.Output(os); return os; } \
friend istream& operator >> (istream& is, CLASSTYPE_XX& x) \
        { x.Input(is); return is; }
```

Definition at line 10 of file streamoper.h.

5.115 top_process.h File Reference

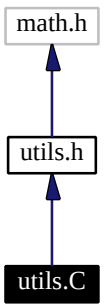
This graph shows which files directly or indirectly include this file:



5.116 utils.C File Reference

```
#include "utils.h"
```

Include dependency graph for utils.C:



Variables

- const char **sccs_ident** [] = "@(#)utils.cpp 2.3 (02/23/98)"
- const int **bit_count_array** [256]

5.116.1 Variable Documentation

5.116.1.1 const int bit_count_array[256]

Initial value:

```
{
0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8,
}
```

Definition at line 5 of file utils.C.

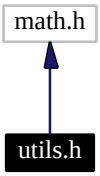
5.116.1.2 const char sccs_ident[] = "@(#)utils.cpp 2.3 (02/23/98)" [static]

Definition at line 3 of file utils.C.

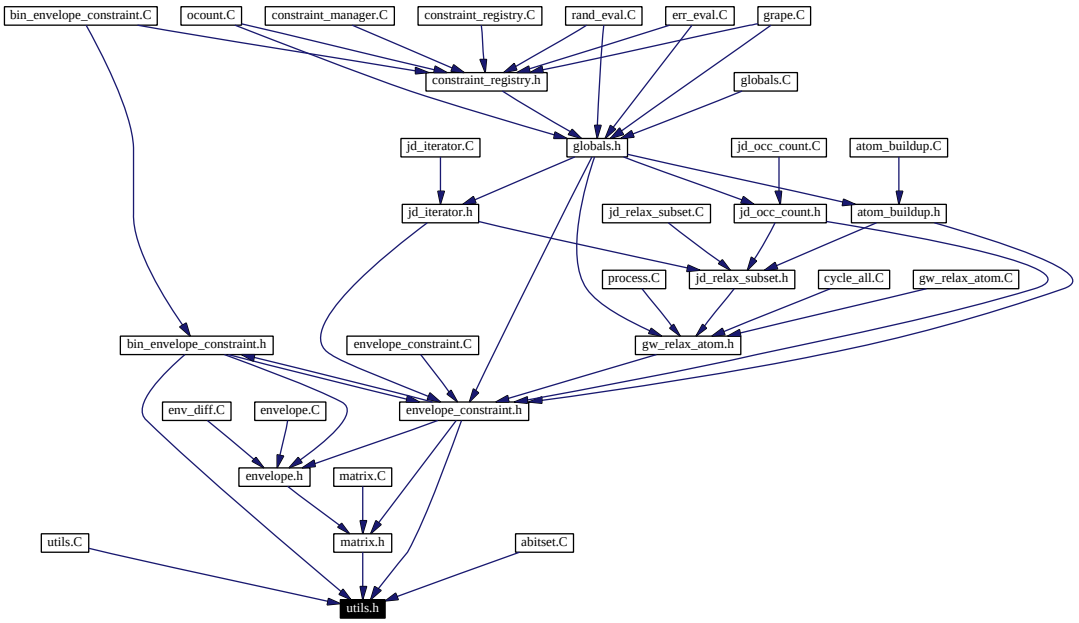
5.117 utils.h File Reference

`#include <math.h>`

Include dependency graph for utils.h:



This graph shows which files directly or indirectly include this file:



Defines

- #define **ERROR**(X)

Functions

- void **util_swap** (T &t1, T &t2)
- T **util_max** (const T &t1, const T &t2)
- T **util_min** (const T &t1, const T &t2)
- T **bound** (const T &t, const T &t1, const T &t2)
- bool **bounded** (const T &t, const T &t1, const T &t2)
- bool **bounds_compare** (const T &t, const T &t1, const T &t2)
- double **mod_dist** (double x, double y, double m)
- int **bit_count** (unsigned char uc)
- int **odd** (int n)
- T **sq** (T &t1)

Variables

- const int **bit_count_array** [256]
- const double **sqrt2**
- const double **sqrt2_2**
- const double **sqrt3**
- const double **sqrt3_2**
- const double **sqrt5**
- const double **sqrt5_2**
- const double **pi**
- const double **pi2**
- const double **pi_2**
- const double **o_sqrt2**
- const double **o_sqrt2_2**
- const double **o_sqrt3**
- const double **o_sqrt3_2**
- const double **o_pi**
- const double **o_pi2**
- const double **o_pi_2**
- const double **rad_to_deg**
- const double **deg_to_rad**

5.117.1 Define Documentation

5.117.1.1 #define ERROR(X)

Value:

```
cerr << "Error: " << __FILE__ << " (" << __LINE__ << ')' \
     << "\t" #X " = " << (X) \
     << endl;
```

Definition at line 133 of file utils.h.

5.117.2 Function Documentation

5.117.2.1 int bit_count (unsigned char *uc*) [inline]

Definition at line 88 of file utils.h.

Referenced by ABitSet::Count().

```
00089 {
00090         return bit_count_array[uc];
00091
```

5.117.2.2 T bound (const T & *t*, const T & *t1*, const T & *t2*) [static]

Definition at line 45 of file utils.h.

```
00046 {
00047         return util_min(util_max(t1,t2), util_max(util_min(t1,t2),t));
00048
```

5.117.2.3 bool bounded (const T & *t*, const T & *t1*, const T & *t2*) [static]

Definition at line 52 of file utils.h.

```
00053 {
00054         return (t <= util_max(t1,t2)) && (t >= util_min(t1,t2));
00055
```

5.117.2.4 bool bounds_compare (const T & *t*, const T & *t1*, const T & *t2*) [static]

Definition at line 59 of file utils.h.

```
00060 {
00061         return
00062                 (t > util_max(t1,t2))
00063                         ? +1
00064                         : ((t < min(t1,t2))
00065                                 ? -1
00066                                 : 0);
00067
```

5.117.2.5 double mod_dist (double *x*, double *y*, double *m*) [inline]

Definition at line 70 of file utils.h.

```
00071 {
00072         double  d = x-y;
00073         if (m == 0.0)
00074                 return d<0.0 ? -d : d;
00075
00076         if (d<0.0)
00077                 d = -d;
00078
00079         const double  p = fmod(d, m);
00080         const double  n = fmod(-p+m, m);
00081         return util_min(p,n);
00082
```

5.117.2.6 int odd (int *n*) [inline]

Definition at line 139 of file utils.h.

Referenced by Matrix::det().

```
00140 {
00141   return (n % 2);
00142
00143
```

5.117.2.7 T sq (T & *t1*) [static]

Definition at line 147 of file utils.h.

Referenced by Matrix::findEigenVectors().

```
00148 {
00149   return (t1 * t1);
00150
```

5.117.2.8 T util_max (const T & *t1*, const T & *t2*) [static]

Definition at line 31 of file utils.h.

Referenced by bound(), bounds_compare(), and BinEnvelopeConstraint::computeStress().

```
00032 {
00033         return t1>t2 ? t1 : t2;
00034
```

5.117.2.9 T util_min (const T & *t1*, const T & *t2*) [static]

Definition at line 38 of file utils.h.

Referenced by ABitSet::Resize(), bound(), bounded(), mod_dist(), ABitSet::operator &=(), ABitSet::operator-=(), ABitSet::operator^=(), and ABitSet::operator|=().

```
00039 {
00040         return t1<t2 ? t1 : t2;
00041
```

5.117.2.10 void util_swap (T & *t1*, T & *t2*) [static]

Definition at line 22 of file utils.h.

Referenced by ABitSet::Clear(), ABitSet::Invert(), ABitSet::Set(), Matrix::swapCols(), and Matrix::swapRows().

```
00023 {
00024         T t = t1;
00025         t1 = t2;
00026         t2 = t;
00027
```

5.117.3 Variable Documentation

5.117.3.1 const int bit_count_array[256]

Initial value:

```
{
0, 1, 1, 2, 1, 2, 2, 3, 1, 2, 2, 3, 2, 3, 3, 4,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
1, 2, 2, 3, 2, 3, 3, 4, 2, 3, 3, 4, 3, 4, 4, 5,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
2, 3, 3, 4, 3, 4, 4, 5, 3, 4, 4, 5, 4, 5, 5, 6,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
3, 4, 4, 5, 4, 5, 5, 6, 4, 5, 5, 6, 5, 6, 6, 7,
4, 5, 5, 6, 5, 6, 6, 7, 5, 6, 6, 7, 6, 7, 7, 8,
}
```

Definition at line 5 of file utils.C.

5.117.3.2 const double deg_to_rad

Initial value:

```
0.017453292519943295769236907684886127134428718
```

Definition at line 129 of file utils.h.

5.117.3.3 const double o_pi

Initial value:

```
0.31830988618379067153776752674502872406891929 1
```

Definition at line 120 of file utils.h.

5.117.3.4 const double o_pi2

Initial value:

```
0.15915494309189533576888376337251436203445964
```

Definition at line 122 of file utils.h.

5.117.3.5 const double o_pi_2

Initial value:

```
0.63661977236758134307553505349005744813783858 2
```

Definition at line 124 of file utils.h.

5.117.3.6 const double o_sqrt2

Initial value:

```
0.70710678118654752440084436210484903928483593 7
```

Definition at line 112 of file utils.h.

5.117.3.7 const double o_sqrt2_2

Initial value:

```
1.41421356237309504880168872420969807856967187 5
```

Definition at line 114 of file utils.h.

5.117.3.8 const double o_sqrt3

Initial value:

```
0.57735026918962576450914878050195745564760175 1
```

Definition at line 116 of file utils.h.

5.117.3.9 const double o_sqrt3_2

Initial value:

```
1.15470053837925152901829756100391491129520350 2
```

Definition at line 118 of file utils.h.

5.117.3.10 const double pi

Initial value:

```
3.14159265358979323846264338327950288419716939 9
```

Definition at line 105 of file utils.h.

5.117.3.11 const double pi2

Initial value:

```
6.28318530717958647692528676655900576839433879 8
```

Definition at line 107 of file utils.h.

5.117.3.12 const double pi_2

Initial value:

```
1.57079632679489661923132169163975144209858469 9
```

Definition at line 109 of file utils.h.

5.117.3.13 const double rad_to_deg

Initial value:

```
57.29577951308232087679815481410517033240547246
```

Definition at line 127 of file utils.h.

5.117.3.14 const double sqrt2

Initial value:

```
1.41421356237309504880168872420969807856967187 5
```

Definition at line 93 of file utils.h.

5.117.3.15 const double sqrt2_2

Initial value:

```
0.70710678118654752440084436210484903928483593 7
```

Definition at line 95 of file utils.h.

5.117.3.16 const double sqrt3

Initial value:

```
1.73205080756887729352744634150587236694280525 3
```

Definition at line 97 of file utils.h.

5.117.3.17 const double sqrt3_2

Initial value:

```
0.86602540378443864676372317075293618347140262 6
```

Definition at line 99 of file utils.h.

5.117.3.18 const double sqrt5

Initial value:

```
2.23606797749978969640917366873127623544061835 9
```

Definition at line 101 of file utils.h.

5.117.3.19 const double sqrt5_2

Initial value:

```
1.11803398874989484820458683436563811772030917 9
```

Definition at line 103 of file utils.h.