A.M.B.R.O.S.I.A.

1.0

Generated by Doxygen 1.8.3.1

Sat May 25 2013 22:17:43

Contents

Chapter 1

AMBROSIA Software

Author

Alexander V. Popov apopov@niboch.nsc.ru

1.1 Introduction

AMBROSIA (**A**mateur **M**odeling of **B**iopolymers: **R**estoration, **O**ptimization, **S**olvation & **I**nitial **A**nalysis) is a molecular dynamics modeling tool. It is cross-platform (Win32, Linux) software supporting both 32-bit and 64-bit architectures, and is optimized for modern multi-core CPUs. AMBROSIA performs the following tasks:

- biopolymer structure refinement (restoration of hydrogens and heavy sidechain atoms);
- disulfide bond detection;
- charge equilibration;
- structure optimization using energy minimization.

The following tasks are under development:

- explicit solvation;
- solvent-accessible surface calculation;
- structure optimization using molecular dynamics simulation;
- global conformational analsys (RMSD/RMSF, free energy, etc.).

1.2 Force Field

The program is based on the AMBER **ff99** force field with some parametric (ff99SB, parmbsc0) and functional form (improper, hydrogen bonding) modifications. AMBROSIA force field has the following general form:

$$E = E_{bond} + E_{angle} + E_{torsion} + E_{improper} + E_{nonb} + E_{special}$$

Non-bonded energy comprises the hydrophobic (van der Waals), electrostatic, hydrogen bonding and implicit solvation interactions:

$$E_{nonb} = E_{vdW} + E_{coulomb} + E_{H-bond} + E_{solv}$$

Special [energy] term is actually an artificial term used to restrain atomic positions or interatomic distances to given values.

1.2.1 Bond energy

Covalent bonds are modeled as rigid harmonic oscillators with the following formulation:

$$E_{bond} = \sum K_{hb}(b_{ij} - b_{ij0})^2$$

where b_{ij} is a distance between atoms i and j connected with a covalent bond, and b_{ij0} is an equilibrium bond length. The summation runs over all covalent bonds in the model.

The harmonic constant is defined in a configuration file (amber99.cfg). Please note that AMBER defines it as the force constant scaled by two. AMBER bond parameters are fully compatible with AMBROSIA.

1.2.2 Bond angle energy

Covalent bond angles are modeled as rigid harmonic oscillators with the following formulation:

$$E_{angle} = \sum K_{ha}(\theta_{ijk} - \theta_{ijk0})^2$$

where θ_{ijk} is an angle between atoms i, j and k connected with covalent bonds $j-i$ and $j-k$, and θ_{ijk0} is an equilibrium angle value. The summation runs over all covalent bond angles in the model.

The harmonic constant is defined in a configuration file (amber99.cfg). Please note that AMBER defines it as the force constant scaled by two. AMBER bond angle parameters are fully compatible with AMBROSIA.

1.2.3 Torsion energy

Covalent bond torsion energy is usually decomposed into fourier series:

$$E_{torsion} = \sum\sum_n \frac{K_{ht}}{s}(1 + cos(n\omega - \phi))$$

where ω is a dihedral angle defined by atoms i, j, k and l. Number of fourier series (n), barrier scale (s), and phase angle (ϕ) are defined in a configuration file. The summation runs over all fourier series defined for the torsion angle, and for all torsion angles in the model. If all atoms comprised by the torsion, belong to a planar ring structure, the harmonic constant is not divided by the barrier scale s.

The harmonic constant is also defined in a configuration file (amber99.cfg). Please note that AMBER defines it as the force constant scaled by two. AMBER torsion angle parameters (including non-degenerate phase angles) are fully compatible with AMBROSIA.

1.2.4 Improper torsion energy

Impropers are used to keep certain dihedral structures in a well-known conformation (e.g. planar aromatic rings). Alike bond angles, impropers are modeled as rigid harmonic oscillators with the following formulation:

$$E_{improper} = \sum K_{hi}(\xi_{ijkl} - \xi_{ijkl0})^2$$

where ξ_{ijkl} is a dihedral angle in a tetrahedron defined by atoms i, j, k and l with the top atom j, and ξ_{ijkl0} is an equilibrium dihedral angle value. The summation runs over all improper angles in the model.

The harmonic constant is defined in a configuration file (amber99.cfg). Since AMBER doesn't treat improper dihedrals separately, the functional form of the improper energy is derived from GROMOS. Please note that ring impropers are created automatically using information from the topology, with the default harmonic constant of 40.0 kcal/mol$*$A^2 . There is no need to define any impropers for aromatic rings, since they are implicitly described in the topology.

1.2.5 Non-bonded intraction energy

Non-bonded interactions comprise two primary and two secondary interactions. The primary interactions are:

- hydrophobic (van der Waals) interaction;
- electrostatic (Coulomb) interaction.

The secondary (and actually optional) interactions include:

- hydrogen bonding interaction;
- interaction with an implicit solvent.

The summation runs in principle over all the non-bonded atomic pairs within certain cut-off radii (except hydrogen bonding interactions, that are summed over triplets). 1-4 neighbours have special scales applied to van der Waals and electrostatic energy and forces: 0.5 for van der Waals, and 1/1.2 for electrostatics.

Van der Waals interaction has a functional form of 12-6 potential with a soft switch function:

$$E_{vdW} = \sum E_{ij} (\frac{R_{ij}^{12}}{r_{ij}^{12}} - 2\frac{R_{ij}^{6}}{r_{ij}^{6}})$$

where r_{ij} is a pairwise distance between atoms i and j, $E_{ij} = \sqrt{E_i E_j}$, $R_{ij} = R_i + R_j$, E_i is a potential well depth for atom i, and R_i is a van der Waals radius of atom i. Both van der Waals parameters E_i and R_i were derived from AMBER. Switch function has the functional form of that utilized in NAMD:

$$F_{switch} = \frac{(R_c^2 - r_{ij}^2)^2 (R_c^2 + 2r_{ij}^2 - 3R_s^2)}{(R_c^2 - R_s^2)^3}$$

where R_c is a short cut-off radius, and R_s is a switch radius ($R_s <= R_c$; note that if $R_s = R_c$, the switch function is not applied).

Electrostatic interaction has a form if Coulomb law with reaction field correction:

$$E_{coulomb} = \frac{1}{Dr_{ij}} q_i q_j (\frac{1}{r_{ij}} - \frac{C_{rf} r_{ij}^2}{R_c^3} - \frac{1 - 0.5C_{rf}}{R_c})$$
$$C_{rf} = \frac{(2-2\varepsilon)(1+\kappa R_c) - \varepsilon(\kappa R_c)^2}{(1+2\varepsilon)(1+\kappa R_c) + \varepsilon(\kappa R_c)^2}$$

where r_{ij} is a pairwise distance between atoms i and j, R_c is a long cut-off radius, ε is a relative dielectic permittivity (80.0 for water), κ is a Debye radius, and D is a distance-dependent dielectric constant scale. By default, it is equal to 1.0, that corresponds a simple linear distant-dependent constant. When $D = 0$, the dielectric constant doesn't depend on the distance.

Reaction field correction is a common practice for taking into account screening effects; for example, it is utilized in GROMOS.

AMBROSIA offers two slightly different models of **hydrogen bonding** energy: 12-6 and 12-8. The first form has a longer range than the second. The default model is 12-8.

$$E_{H-bond-126} = S_H \sum E_H (\frac{R_H^{12}}{r_{ij}^{12}} - 2\frac{R_H^{6}}{r_{ij}^{6}}) e^{-\frac{(cos\theta - cos\theta_0)^2}{\sigma^2}}$$
$$E_{H-bond-128} = S_H \sum E_H (\frac{R_H^{12}}{r_{ij}^{12}} - 1.5\frac{R_H^{8}}{r_{ij}^{8}}) e^{-\frac{(cos\theta - cos\theta_0)^2}{\sigma^2}}$$

where r_{ij} is a pairwise distance between hydrogen atom and acceptor atom, S_H is a global hydrogen bonding energy scale (1.0 by default; can be increased when hydrogen bonding is essential for stability of the structure), R_H and E_H are parametrized values for equilibrium distance and well depth, θ is a triplet angle formed by donor, hydrogen and acceptor atoms, θ_0 is an ideal angle (π radians), and σ is a constant ($cos\frac{5\pi}{6} - cos\pi$).

These models use common parameters derived from BioPASED software package.

Implicit **solvation** energy term depends on a solvation model selected and is described elsewhere.

1.2.6 Special energy

AMBROSIA supports two types of restraining functions:

- positional restraints;
- distant restraints.

Positional restraints allow to keep atoms softly approaching their initial positions. The potential has the following functional form:

$$E_{rp} = \sum K_{hr}(r_i - r_{i0})^2$$

where r_i is the current position of an atom i, and r_{i0} is an initial position of the atom (before any optimizations). The summation runs over all positionally restrained atoms of the model.

Distant restraints allow to keep two different atoms within the distance d. AMBROSIA utilizes a soft form of the potential (unlike the covalent-bond potential):

$$E_{rd} = \sum \frac{K_{hr}}{2d_{ij0}^2}(d_{ij}^2 - d_{ij0}^2)^2$$

where d_{ij} is the distance between atoms i and j, and d_{ij0} is the equilibrium distance. The summation runs over all pairs of distantly restrained atoms of the model.

The harmonic constant K_{hr} of a restraint is, as usual, a force constant scaled by two.

1.3 Solvation

TODO

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

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

Chapter 3

Class Index

3.1 Class List

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

Chapter 4

File Index

4.1 File List

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

Chapter 5

Class Documentation

5.1 AtomSearchFunctor Class Reference

Atom search functor.

Public Member Functions

- AtomSearchFunctor (const char *const title)

    Constructor.
- bool operator() (const cAtom &at)

    Compare operator.

5.1.1 Detailed Description

Atom search functor.

5.1.2 Constructor & Destructor Documentation

5.1.2.1 AtomSearchFunctor::AtomSearchFunctor (const char *const *title*) `[inline]`

Constructor.

Parameters

title	: title of the atom to search.

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

- src/md_model.h

5.2 AtomSortFunctor Class Reference

Atom sort functor.

Public Member Functions

- bool operator() (const cAtom &at1, const cAtom &at2)

Compare operator.

5.2.1 Detailed Description

Atom sort functor.

Sort atoms by chain, then by residuenumber, then by sortnumber.

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

- src/md_model.h

5.3 cAngle Struct Reference

Single covalent angle of a model.

Public Attributes

- int mAtomIndices [3]

  Indices of atoms comprising the angle.
- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol*rad^2)$), derived from cAngleInfo.
- vec_t mTheta

  Equilibrium angle value (in radians), derived from cAngleInfo.

5.3.1 Detailed Description

Single covalent angle of a model.

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

- src/md_model.h

5.4 cAngleInfo Struct Reference

Description of a valent angle.

Public Attributes

- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol*rad^2)$).
- vec_t mTheta

  Equilibrium bond angle (in radians).

5.4.1 Detailed Description

Description of a valent angle.

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

- src/md_config.h

5.5 cAtom Struct Reference

Single atom of a model.

Public Attributes

- struct stPhysAtom ∗ mpPhysAtom

  Pointer to physics properties (NULL until topology is complete).
- cConnectivity ∗ mpConnectivity

  Pointer to connectivity information (NULL until topology is complete).
- cSolvParms ∗ mpSolvation

  Pointer to solvation parameters (NULL until topology is complete and some solvation model is enabled).
- cQEqParms ∗ mpQEq

  Pointer to qEq parameters (NULL until topology is complete and autoQ is enabled).
- int mFlags

  Atomic flags (see AF_xxx defines).
- vec_t ∗ mCurrentPosition

  Pointer to physics position (NULL until topology is complete).
- vec_t ∗ mCurrentForce

  Pointer to physics force (NULL until topology is complete).
- vec3_t mOriginalPosition

  Original position of the atom (in angstroms).
- vec_t mPosRestraintHarmConst

  Harmonic constant for position restraining (in $kcal/(mol * angstrom^2)$).
- int mFFCode

  Force field code for the atom.
- int mVdWCode

  Van der Waals code for the atom (for pair table lookup).
- int mHBCode

  Hydrogen bond code for the atom (for pair table lookup).
- int mSortNumber

  Sort number of the atom (used by sorting procedures).
- int mResidueNumber

  Serial number of the residue (one-based index).
- int mChainNumber

  Serial number of the chain (one-based index).
- int mResidueSequenceNumber

  Sequence number of the residue (matches PDB data).
- const cResidueAtom ∗ mpResidueAtom

  Pointer to residue atom description.
- char mAtomSymbol [4]

  Atom symbol.
- char mAtomTitle [8]

  Atom title.
- char mResidueTitle [8]

  Residue title.

5.5.1 Detailed Description

Single atom of a model.

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

- src/md_model.h

5.6 cAtomInfo Struct Reference

Description of an atom.

Public Attributes

- vec_t mMassReciprocal

  Resiprocal mass (in a.e.m.).
- vec_t mRadius

  Atomic radius (in angstroms).
- vec_t mXi

  Electronegativity (qEq).
- vec_t mJ0

  Self-repulsion (qEq).

5.6.1 Detailed Description

Description of an atom.

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

- src/md_config.h

5.7 cBond Struct Reference

Single covalent bond of a model.

Public Attributes

- int mAtomIndices [2]

  Indices of atoms comprising the bond.
- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol * angstrom^2)$), derived from cBondInfo.
- vec_t mLength

  Equilibrium bond length (in angstroms), derived from cBondInfo.

5.7.1 Detailed Description

Single covalent bond of a model.

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

- src/md_model.h

5.8 cBondInfo Struct Reference

Description of a bond.

Public Attributes

- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol * angstrom^2)$).
- vec_t mLength

  Equilibrium bond length (in angstroms).

5.8.1 Detailed Description

Description of a bond.

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

- src/md_config.h

5.9 cConfig Class Reference

This object incapsulates persistant molecular dynamics configuration.

Public Types

- typedef std::map< const char
  *, int, StringCompareFunctor > FFCodeMap

  Type for a map of force field atom name and code.
- typedef std::map< const char
  *, cAtomInfo,
  StringCompareFunctor > AtomInfoMap

  Type for a map of cAtomInfo objects.
- typedef std::map< const char
  *, cResidueInfo,
  StringCompareFunctor > ResidueInfoMap

  Type for a map of cResidueInfo objects.
- typedef std::map< const char
  *, const char
  *, StringCompareFunctor > ResidueAliasMap

  Type for a map of cResidueInfo objects.
- typedef std::map< std::pair
  < int, int >, cBondInfo > BondInfoMap

  Type for a map of cBondInfo objects.
- typedef std::map< std::pair
  < int, std::pair< int, int >
  >, cAngleInfo > AngleInfoMap

  Type for a map of cAngleInfo objects.
- typedef std::map< std::pair
  < std::pair< int, int >
  , std::pair< int, int >
  >, cImproperInfo > ImproperInfoMap

Type for a map of cImproperInfo objects.

- typedef std::list< cTorsionInfo > TorsionInfoList

  Type for a list of cTorsionInfo objects (list is used instead of map, because we support "any" type of atom).
- typedef std::map< int, cVdWInfo > VdWInfoMap

  Type for a map of cVdWInfo objects.
- typedef std::map< std::pair
  < int, int >, cHBInfo > HBInfoMap

  Type for a map of cHBInfo objects.
- typedef std::vector< int > HBCodeVector

  Type for a list of hydrogen/acceptor codes.
- typedef std::map< int,
  cSolvGSInfo > SolvGSInfoMap

  Type for a map of cSolvGSInfo objects.
- typedef std::vector
  < cPosRestraintInfo > PosRestrVector

  Type for a list of positional restraints.
- typedef std::vector
  < cDistRestraintInfo > DistRestrVector

  Type for a list of distant restraints.

Public Member Functions

- void LoadConfig (const char ∗const name)

  Loads and parses configuration file.
- void LoadUserConfig (void)

  Loads and parses user-defined configuration file.
- void LoadParams (void)

  Loads and parses file with simulation parameters.
- void Clear (void)

  Clears the config, frees all data allocated.
- int LookupFFCode (const char ∗const title)

  Get force field code for a given force field title.
- const char ∗ LookupFFTitle (const int code)

  Get force field title for a given force field code.
- int LookupSFFCode (const char ∗const title)

  Get solvent force field code for a given solvent force field title.
- const char ∗ LookupSFFTitle (const int code)

  Get solvent force field title for a given solvent force field code.
- const cAtomInfo ∗ LookupAtomInfo (const char ∗const symbol)

  Get description of an atom for the atomic symbol.
- const cBondInfo ∗ LookupBondInfo (int ffCode1, int ffCode2)

  Get description of a bond for given atoms.
- const cAngleInfo ∗ LookupAngleInfo (int ffCode1, int ffCode2, int ffCode3)

  Get description of an angle for given atoms.
- const cImproperInfo ∗ LookupImproperInfo (int ffCodeI, int ffCodeJ, int ffCodeK, int ffCodeL)

  Get description of an improper torsion angle for given atoms.
- const cTorsionInfo ∗ LookupTorsionInfo (int ffCodeI, int ffCodeJ, int ffCodeK, int ffCodeL)

  Get description of a proper torsion angle for given atoms.
- const cVdWInfo ∗ LookupVdWInfo (int ffCode)

  Get description of van der Waals parameters for a given atom.
- const cHBInfo ∗ LookupHBInfo (int ffCodeH, int ffCodeA)

Get description of hydrogen bond parameters for a given atom pair.

- bool IsHBHydrogen (int ffCode)

  Return whether an atom is a hydrogen that can form a hydrogen bond.
- bool IsHBAcceptor (int ffCode)

  Return whether an atom is an acceptor that can form a hydrogen bond.
- const HBCodeVector & GetHBHydrogens (void)

  Return list of known hydrogen-bonding hydrogen FF codes.
- const HBCodeVector & GetHBAcceptors (void)

  Return list of known hydrogen-bonding acceptor FF codes.
- const cSolvGSInfo ∗ LookupSolvGSInfo (int sffCode)

  Get description of Gaussian solvation parameters for a given atom pair.
- const cResidueLocation ∗ LookupResidueInfo (const char ∗const name, eResidueLocation loc)

  Get pointer to a structure describing the residue in a given location.
- bool CheckResidueInfo (const char ∗const name)

  Check whether a named residue is defined in the topology library.
- const cParameters & Parameters (void)

  Helper function to access simulation parameters.
- const PosRestrVector & PositionalRestraints (void)

  Helper function to access positional restraints.
- const DistRestrVector & DistantRestraints (void)

  Helper function to access distant restraints.

Static Public Member Functions

- static void SetUserConfigName (const char ∗name)

  Sets name of a user-defined configuration file.
- static const char ∗ UserConfigName (void)

  Returns a name of a user-defined configuration file.
- static void SetParmsFileName (const char ∗name)

  Sets name of a file with simulation parameters.
- static const char ∗ ParmsFileName (void)

  Returns a name of a file with simulation parameters.
- static const char ∗ LocationToString (eResidueLocation loc)

  Returns a string for a given location index (useful for logging).

5.9.1 Detailed Description

This object incapsulates persistant molecular dynamics configuration.

The following parameters are defined in config:

1. atomic properties (mass and radius);
2. covalent bond properties.

5.9.2 Member Function Documentation

5.9.2.1 void cConfig::SetUserConfigName (const char ∗ *name*) `[static]`

Sets name of a user-defined configuration file.

Default is empty (no user-defined file).

Parameters

name	: new configuration file name.

5.9.2.2 static const char∗ cConfig::UserConfigName (void) [inline], [static]

Returns a name of a user-defined configuration file.

Returns

The name of a user-defined configuration file.

5.9.2.3 void cConfig::SetParmsFileName (const char ∗ *name*) [static]

Sets name of a file with simulation parameters.

Default is DEFAULT_PARMS_FILENAME.

Parameters

name	: new file name.

5.9.2.4 static const char∗ cConfig::ParmsFileName (void) [inline], [static]

Returns a name of a file with simulation parameters.

Returns

The name of a file with simulation parameters.

5.9.2.5 const char ∗ cConfig::LocationToString (eResidueLocation *loc*) [static]

Returns a string for a given location index (useful for logging).

Parameters

loc	: location index.

Returns

The name of the location.

5.9.2.6 void cConfig::LoadConfig (const char ∗const *name*)

Loads and parses configuration file.

If any config data already exists, new data is appended, overriding old data when necessary.

Parameters

name	: config file name.

5.9.2.7 void cConfig::LoadUserConfig (void)

Loads and parses user-defined configuration file.

File name is defined with SetUserConfigName. If the name is empty, nothing is loaded.

5.9.2.8 void cConfig::LoadParams (void)

Loads and parses file with simulation parameters.

File name is defined with SetParmsFileName.

5.9.2.9 int cConfig::LookupFFCode (const char ∗const *title*)

Get force field code for a given force field title.

Parameters

title	: force field title of the atom ("HW", "OW", "N∗", etc.), must be uppercase.

Returns

Force field code identifier.

5.9.2.10 const char ∗ cConfig::LookupFFTitle (const int *code*)

Get force field title for a given force field code.

Parameters

code	: force field code identifier.

Returns

Force field title of the atom ("HW", "OW", "N∗", etc.), uppercase.

5.9.2.11 int cConfig::LookupSFFCode (const char ∗const *title*)

Get solvent force field code for a given solvent force field title.

Parameters

title	: solvent force field title of the atom, must be uppercase.

Returns

Solvent force field code identifier.

5.9.2.12 const char∗ cConfig::LookupSFFTitle (const int *code*)

Get solvent force field title for a given solvent force field code.

Parameters

code	: solvent force field code identifier.

Returns

Solvent force field title of the atom, uppercase.

5.9.2.13 const **cAtomInfo** ∗ cConfig::LookupAtomInfo (const char ∗const *symbol*)

Get description of an atom for the atomic symbol.

Parameters

symbol	: atomic symbol ("H", "C", "N", etc.), must be uppercase.

Returns

Constant pointer to structure describing the atom.

5.9.2.14 const **cBondInfo** ∗ cConfig::LookupBondInfo (int *ffCode1,* int *ffCode2*)

Get description of a bond for given atoms.

Parameters

ffCode1	: force field code for the first atom.
ffCode2	: force field code for the second atom.

Returns

Constant pointer to structure describing the bond.

5.9.2.15 const **cAngleInfo** ∗ cConfig::LookupAngleInfo (int *ffCode1,* int *ffCode2,* int *ffCode3*)

Get description of an angle for given atoms.

Parameters

ffCode1	: force field code for the first atom.
ffCode2	: force field code for the second atom.
ffCode3	: force field code for the third atom.

Returns

Constant pointer to structure describing the angle.

5.9.2.16 const **cImproperInfo** ∗ cConfig::LookupImproperInfo (int *ffCodeI,* int *ffCodeJ,* int *ffCodeK,* int *ffCodeL*)

Get description of an improper torsion angle for given atoms.

Parameters

ffCodeI	: force field code for the atom I.
ffCodeJ	: force field code for the atom J (center).
ffCodeK	: force field code for the atom K.
ffCodeL	: force field code for the atom L (terminator).

Returns

Constant pointer to structure describing the improper.

5.9.2.17 const cTorsionInfo ∗ cConfig::LookupTorsionInfo (int *ffCodeI,* int *ffCodeJ,* int *ffCodeK,* int *ffCodeL*)

Get description of a proper torsion angle for given atoms.

Parameters

ffCodeI	: force field code for the atom I.
ffCodeJ	: force field code for the atom J.
ffCodeK	: force field code for the atom K.
ffCodeL	: force field code for the atom L.

Returns

Constant pointer to structure describing the torsion.

5.9.2.18 const cVdWInfo ∗ cConfig::LookupVdWInfo (int *ffCode*)

Get description of van der Waals parameters for a given atom.

Parameters

ffCode	: force field code for the atom.

Returns

Constant pointer to structure describing van der Waals parameters.

5.9.2.19 const cHBInfo ∗ cConfig::LookupHBInfo (int *ffCodeH,* int *ffCodeA*)

Get description of hydrogen bond parameters for a given atom pair.

Parameters

ffCodeH	: force field code for the hydrogen atom.
ffCodeA	: force field code for the acceptor atom.

Returns

Constant pointer to structure describing hydrogen bond parameters.

5.9.2.20 bool cConfig::IsHBHydrogen (int *ffCode*)

Return whether an atom is a hydrogen that can form a hydrogen bond.

Parameters

ffCode	: force field code for the atom.

Returns

True or false.

5.9.2.21 bool cConfig::IsHBAcceptor (int *ffCode*)

Return whether an atom is an acceptor that can form a hydrogen bond.

Parameters

ffCode	: force field code for the atom.

Returns

True or false.

5.9.2.22 const HBCodeVector& cConfig::GetHBHydrogens (void) [inline]

Return list of known hydrogen-bonding hydrogen FF codes.

Returns

HBCodeVector.

5.9.2.23 const HBCodeVector& cConfig::GetHBAcceptors (void) [inline]

Return list of known hydrogen-bonding acceptor FF codes.

Returns

HBCodeVector.

5.9.2.24 const cSolvGSInfo ∗ cConfig::LookupSolvGSInfo (int *sffCode*)

Get description of Gaussian solvation parameters for a given atom pair.

Parameters

sffCode	: force field code for the atom.

Returns

Constant pointer to structure describing solvation parameters.

5.9.2.25 const cResidueLocation ∗ cConfig::LookupResidueInfo (const char ∗const *name*, eResidueLocation *loc*)

Get pointer to a structure describing the residue in a given location.

Parameters

name	: name of the residue ("GLY", "ALA", "G", etc.), must be uppercase.
loc	: location of the residue.

Returns

Constant pointer to a structure describing the residue in a given location.

5.9.2.26 bool cConfig::CheckResidueInfo (const char *const *name*)

Check whether a named residue is defined in the topology library.

Parameters

name	: name of the residue ("GLY", "ALA", "G", etc.), must be uppercase.

Returns

True (exists) or false (doesn't exist).

5.9.2.27 const cParameters& cConfig::Parameters (void) [inline]

Helper function to access simulation parameters.

Returns

Constant reference to structure describing simulation parameters.

5.9.2.28 const PosRestrVector& cConfig::PositionalRestraints (void) [inline]

Helper function to access positional restraints.

Returns

Constant reference to vector of positional restraints.

5.9.2.29 const DistRestrVector& cConfig::DistantRestraints (void) [inline]

Helper function to access distant restraints.

Returns

Constant reference to vector of distant restraints.

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

- src/md_config.h
- src/md_config.cpp

5.10 cConnectivity Struct Reference

Connectivity information for an atom.

Public Attributes

- IntArray mNeighbours_12_13

  List of first and second neighbours (1-2 and 1-..-3).
- IntArray mNeighbours_14

  List of third neighbours (1-..-..-4).
- IntArray mNeighbours_12H

  List of first neighbour hydrogens.

5.10.1 Detailed Description

Connectivity information for an atom.

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

- src/md_model.h

5.11 cDataFormat$<$ T, U $>$ Struct Template Reference

Abstract data format.

Public Types

- typedef T DataElement

  Data type alias.
- typedef U DataHeader

  Header type alias.
- typedef std::vector$<$ T $>$ DataArray

  Array of data type (STL vector).

Public Member Functions

- virtual void ReadFile (const char ∗const filename, DataHeader ∗const header, DataArray ∗const data)=0

  Reads array of data from file.
- virtual void WriteFile (const char ∗const filename, const DataHeader ∗const header, const DataArray ∗const data)=0

  Writes array of data to file.

5.11.1 Detailed Description

template$<$typename T, typename U$>$struct cDataFormat$<$ T, U $>$

Abstract data format.

Template Parameters

T	: data structure of element to work with.
U	: data structure of header to work with.

5.11.2 Member Function Documentation

5.11.2.1 template<typename T, typename U> virtual void **cDataFormat**< T, U >::ReadFile (const char *const *filename*, **DataHeader** *const *header*, **DataArray** *const *data*) `[pure virtual]`

Reads array of data from file.

Parameters

filename	: file name to read.
header	: pointer to header to read in.
data	: pointer to data to read in.

Implemented in cPDBFormat.

5.11.2.2 template<typename T, typename U> virtual void **cDataFormat**< T, U >::WriteFile (const char *const *filename*, const **DataHeader** *const *header*, const **DataArray** *const *data*) `[pure virtual]`

Writes array of data to file.

Parameters

filename	: file name to write.
header	: pointer to header to write out.
data	: pointer to data to write out.

Implemented in cPDBFormat.

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

- src/md_format.h

5.12 cDistRestrain Struct Reference

Single distant restraint of a model.

Public Attributes

- int mAtomIndices [2]

  Indices of atoms comprising the restraint.
- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol*angstrom^2)$).
- vec_t mLengthSquared

  Square of equilibrium interatomic distance (in square angstroms).

5.12.1 Detailed Description

Single distant restraint of a model.

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

- src/md_model.h

5.13 cDistRestraintInfo Struct Reference

Description of distant restraint.

Public Attributes

- int mChain [2]

  Index of the chain.
- int mResidue [2]

  Index of the residue.
- char mAtomTitle [2][8]

  Title of the atom.
- vec_t mHarmConst

  Harmonic constant (in $kcal/(mol * angstrom^2)$).
- vec_t mDistance

  True if restraint should be applied to backbone only.

5.13.1 Detailed Description

Description of distant restraint.

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

- src/md_config.h

5.14 cFindData Struct Reference

Find data for file search funtions.

Public Attributes

- int mAttrib

  Attributes.
- qword mTimeCreate

  Creation time.
- qword mTimeWrite

  Modification time.
- char mFileName [MAX_OSPATH]

  File name (relative to search path).

5.14.1 Detailed Description

Find data for file search funtions.

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

- src/md_common.h

5.15 cHBInfo Struct Reference

Description of H-bond parameters of an atom.

Public Attributes

- vec_t mR

  Rmin/2, in angstroms.
- vec_t mE

  Well depth, in kcal/mol.

5.15.1 Detailed Description

Description of H-bond parameters of an atom.

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

- src/md_config.h

5.16 cHBPair Struct Reference

Hydrogen bonding pair parameters.

Public Attributes

- vec_t mR

  Value of Rmin for the pair.
- vec_t mE

  Value of E for the pair.
- vec_t mA

  Value of A (C12) for the pair.
- vec_t mB

  Value of B (C6 or C8) for the pair.

5.16.1 Detailed Description

Hydrogen bonding pair parameters.

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

- src/md_model.h

5.17 cHBTriplet Struct Reference

Hydrogen bond triplet (D = donor, H = hydrogen, A = acceptor).

Public Attributes

- int mAtomIndices [3]

  Indices of atoms comprising the triplet (D-H..A).
- const cHBPair ∗ mpHBPair

  Pointer to hydrogen bond pair (H..A).

5.17.1 Detailed Description

Hydrogen bond triplet (D = donor, H = hydrogen, A = acceptor).

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

- src/md_model.h

5.18 cImproper Struct Reference

Single improper torsion angle of a model.

Public Attributes

- int mAtomIndices [4]

  Indices of atoms comprising the improper (I-J-K-L).
- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol*rad^2)$), derived from cImproperInfo.
- vec_t mXi

  Equilibrium angle value (in radians), derived from cImproperInfo.

5.18.1 Detailed Description

Single improper torsion angle of a model.

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

- src/md_model.h

5.19 cImproperInfo Struct Reference

Description of an improper torsion angle.

Public Attributes

- vec_t mHarmonicConstant

  Harmonic force constant (in $kcal/(mol*rad^2)$).
- vec_t mXi

  Equilibrium improper angle (in radians).

5.19.1 Detailed Description

Description of an improper torsion angle.

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

- src/md_config.h

5.20 cLog Class Reference

Object responsible for logging and error reporting.

Public Member Functions

- void Close (void)

  Manually close logging.
- void Printf (const char ∗fmt,...)

  Print formatted text both to the log file and stdout.
- void DPrintf (const char ∗fmt,...)

  Print formatted text to the log file.
- void CPrintf (const char ∗fmt,...)

  Print formatted text to the console ONLY.
- void Verbose (const char ∗fmt,...)

  Print formatted text to the log file if verbose mode is set.
- void TPrintf (const char ∗fmt,...)

  Print formatted text to the log file without timestamp prefix.
- void Warning (const char ∗fmt,...)

  Print formatted warning message both to the log file and stdout.
- void Error (const char ∗fmt,...)

  Print formatted error message both to the log file and stdout.
- void Fatal (const char ∗fmt,...)

  Print formatted error message both to the log file and stdout.
- void NewLine (void)

  Print newline to log file.

Static Public Member Functions

- static void SetFileName (const char ∗name)

  Sets name of a log file.
- static const char ∗ FileName (void)

  Returns a name of a log file.
- static void EnableVerboseMode (void)

  Enables verbose output.
- static bool VerboseMode (void)

  Returns whether verbose output is enabled.

5.20.1 Detailed Description

Object responsible for logging and error reporting.

This object is used for the following tasks:

1. printing messages to log file with or without timestamps;
2. printing messages to standard output (console);
3. printing warnings and errors;
4. printing **fatal errors** with a termination of the whole program.

5.20.2 Member Function Documentation

5.20.2.1 void cLog::SetFileName (const char ∗ *name*) [static]

Sets name of a log file.

Default is DEFAULT_LOG_FILENAME.

Parameters

name	: new log file name.

5.20.2.2 static const char∗ cLog::FileName (void) [inline], [static]

Returns a name of a log file.

Returns

The name of a log file.

5.20.2.3 void cLog::Printf (const char ∗ *fmt*, ...)

Print formatted text both to the log file and stdout.

Parameters

fmt	: message / format string.

5.20.2.4 void cLog::DPrintf (const char ∗ *fmt*, ...)

Print formatted text to the log file.

Parameters

fmt	: message / format string.

5.20.2.5 void cLog::CPrintf (const char ∗ *fmt*, ...)

Print formatted text to the console ONLY.

Parameters

fmt	: message / format string.

5.20.2.6 void cLog::Verbose (const char ∗ *fmt*, ...)

Print formatted text to the log file if verbose mode is set.

Parameters

fmt	: message / format string.

5.20.2.7 void cLog::TPrintf (const char ∗ *fmt*, ...)

Print formatted text to the log file without timestamp prefix.

Parameters

fmt	: message / format string.

5.20.2.8 void cLog::Warning (const char ∗ *fmt*, ...)

Print formatted warning message both to the log file and stdout.

This also increments warning counter.

Parameters

fmt	: message / format string.

5.20.2.9 void cLog::Error (const char ∗ *fmt*, ...)

Print formatted error message both to the log file and stdout.

This also increments error counter.

Parameters

fmt	: message / format string.

5.20.2.10 void cLog::Fatal (const char ∗ *fmt*, ...)

Print formatted error message both to the log file and stdout.

This also increments error counter.

Program execution is immediately terminated after this call.

Parameters

fmt	: message / format string.

5.20.2.11 void cLog::NewLine (void)

Print newline to log file.

Just for formatting in a more readable way.

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

- src/md_log.h
- src/md_log.cpp

5.21 cModel Class Reference

Object representing a 3D-model of biopolymer.

Public Types

- typedef std::vector< cAtom > AtomArray

  Type for an array of atoms.
- typedef std::vector< cBond > BondArray

  Type for an array of bonds.
- typedef std::vector< cSSBond > SSBondArray

  Type for an array of S-S bonds.
- typedef std::vector
  < cDistRestrain > DistRestArray

  Type for an array of distant restraints.
- typedef std::vector< cAngle > AngleArray

  Type for an array of angles.
- typedef std::vector< cImproper > ImproperArray

  Type for an array of impropers.
- typedef std::vector< cTorsion > TorsionArray

  Type for an array of torsions.
- typedef std::vector
  < cNonBondedPair > NBPArray

  Type for an array of non-bonded pairs (NBP).
- typedef std::vector< cHBTriplet > HBTArray

  Type for an array of hydrogen bond triplets (HBT).
- typedef std::map< const char
  ∗, cDataFormat< cAtom,
  cModelHeader >
  ∗, StringCompareFunctor > IOMap

  Type for a map of IO objects.

Public Member Functions

- void Load (void)

  Loads a source structure file into memory.
- void Save (const char ∗const filename)

  Saves current snapshot to the structure file.
- void Clear (void)

  Clears the model, frees all data allocated.
- void InitializeParams (void)

  Initialize persistant modelling parameters.
- void BuildTopology (void)

  Builds model's topology.
- void CalcEnergy (bool initial)

  Calculates energy for the structure.
- void Optimize (int nSteps)

  Optimizes the structure using local energy minimization.
- void ProfileReport (void)

  Print performance profiling results to the log file.

Static Public Member Functions

- static void SetFileName (const char ∗name)

  Sets name of a source structure file.
- static const char ∗ FileName (void)

  Returns a name of a source structure file.
- static void SetModelName (const char ∗name)

  Sets name of a model.
- static const char ∗ ModelName (void)

  Returns a name of a model.
- static void EnableProfiling (void)

  Enable performance profiling (debug feature).
- static void InitializeIO (void)

  Initialize IO objects for different data formats.
- static vec_t EnergyMinimizationFunction (vec_t delta)

  Static function to access "CalcDeltaEnergy" member of model object from mathlib.

Friends

- class **cThreadManager**

5.21.1 Detailed Description

Object representing a 3D-model of biopolymer.

This object is used for the following tasks:

1. loading the model from a structure file;
2. saving current model state to a structure file.

5.21.2 Member Function Documentation

5.21.2.1 void cModel::SetFileName (const char ∗ *name*) `[static]`

Sets name of a source structure file.

Default is DEFAULT_MODEL_FILENAME.

Parameters

name	: new source structure file name.

5.21.2.2 static const char∗ cModel::FileName (void) `[inline], [static]`

Returns a name of a source structure file.

Returns

The name of a source structure file.

5.21.2.3 void cModel::SetModelName (const char * *name*) [static]

Sets name of a model.

This is a prefix for all output structure files. Default is empty, that means no prefix is added.

Parameters

name	: new name of a model.

5.21.2.4 static const char* cModel::ModelName (void) [inline], [static]

Returns a name of a model.

Returns

The name of a model.

5.21.2.5 static void cModel::EnableProfiling (void) [inline], [static]

Enable performance profiling (debug feature).

When profiling is enabled, most performance-critical subroutines accumulate their execution time. The statistics is displayed when program terminates.

5.21.2.6 void cModel::InitializeIO (void) [static]

Initialize IO objects for different data formats.

This function should be modified upon implementation of a new data format.

5.21.2.7 vec_t cModel::EnergyMinimizationFunction (vec_t *delta*) [static]

Static function to access "CalcDeltaEnergy" member of model object from mathlib.

This function is not thread-safe.

Parameters

delta	: coordinate offset to apply temporarily to the model.

Returns

The energy of the structure with coordinate offset applied.

5.21.2.8 void cModel::Load (void)

Loads a source structure file into memory.

Structure file type is determined by file name extension, default is **PDB**. This also prepares the structure for further operation.

5.21.2.9 void cModel::Save (const char *const *filename*)

Saves current snapshot to the structure file.

Structure file type is determined by file name extension, default is **PDB**.

Parameters

filename	: name of the structure file to write snapshot to.

5.21.2.10 void cModel::BuildTopology (void)

Builds model's topology.

The function performs the following tasks:

1. restores missing heavy atoms;
2. restores missing hydrogen atoms;
3. initializes arrays of bonds, angles, impropers and torsions. This must be called after the model is loaded.

5.21.2.11 void cModel::CalcEnergy (bool *initial*)

Calculates energy for the structure.

The function performs the following tasks:

1. calculates total energy;
2. writes PDB file with remarks.

Parameters

initial	: true if this is an initial energy calculation.

5.21.2.12 void cModel::Optimize (int *nSteps*)

Optimizes the structure using local energy minimization.

The function performs optimization using conjugate gradients method.

Parameters

nSteps	: maximum number of optimization steps.

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

- src/md_model.h
- src/md_model.cpp
- src/md_model_energy.cpp
- src/md_model_nbp.cpp
- src/md_model_qeq.cpp
- src/md_model_top.cpp

5.22 cModelHeader Struct Reference

Model header (global data).

Public Attributes

- int mNumAtoms

  Number of atoms in a model.
- int mNumHeavyAtoms

  Number of heavy atoms (i.e. not hydrogens) in a model.
- int mNumResidues

  Number of residues in a model.
- int mNumChains

  Number of chains in a model.
- char mComment [8192]

  Header comment.

5.22.1 Detailed Description

Model header (global data).

This is a group of persistant data that can be saved or restored.

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

- src/md_model.h

5.23 cNonBondedPair Struct Reference

Non-bonded pair.

Public Attributes

- int mAtomIndices [2]

  Indices of atoms comprising the pair.
- int mFlags

  Pair flags.
- const cVdWPair $*$ mpVdWPair

  Pointer to VdW pair (can be NULL).

5.23.1 Detailed Description

Non-bonded pair.

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

- src/md_model.h

5.24 cParameters Struct Reference

Simulation parameters that are set up via parameters' file.

Public Attributes

- int mLoadModel

    Defines which model to read (from files that support multiple models, e.g. PDB).
- int mReadHydrogens

    Defines whether to read H-atoms from the source structure, or restore them automatically.
- int mAutoSSBonds

    Automatic detection of S-S bonds.
- vec_t mSSBondDist

    Maximum length of a recognizable S-S bond.
- int mAutoQ

    Automatic (re)calculation of partial charges.
- int mAutoRSN

    Automatic rebase of residue serial numbers.
- int mEnergyCalc

    Calculate energy for the source structure.
- int mEnergyOptimize

    Optimize energy for the source structure.
- int mEnergyOptimizeSteps

    Perform N steps of optimization procedure.
- vec_t mShortCutoffDist

    Van der Waals cut-off distance, in angstroms.
- vec_t mShortSwitchDist

    Van der Waals switch distance, in angstroms.
- vec_t mLongCutoffDist

    Electrostatic cut-off distance, in angstroms.
- vec_t mDistanceDependentDielectricConstant

    Scale of distance-dependent dielectric constant (0 = no distance dependence).
- vec_t mSolventDielectricConstant

    Dielectric constant of the solvent.
- vec_t mDebyeRadius

    Debye radius of the solvent, in angstroms.
- int mHBondModel

    Hydrogen bonding model (either 126 or 128).
- vec_t mHBondScale

    Hydrogen bonding energy scale.
- vec_t mHBondCutoffDist

    Hydrogen bonding cut-off distance, in angstroms.
- char ∗ mSolvationModel

    Solvation model: NULL (none), GS (gaussian), GB (generalized Born).

5.24.1 Detailed Description

Simulation parameters that are set up via parameters' file.

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

- src/md_config.h

5.25 cPDBFormat Class Reference

PDB file format.

Inherits cDataFormat< cAtom, cModelHeader >.

Public Member Functions

- virtual void ReadFile (const char ∗const filename, DataHeader ∗const header, DataArray ∗const data)

  Reads array of data from file.
- virtual void WriteFile (const char ∗const filename, const DataHeader ∗const header, const DataArray ∗const data)

  Writes array of data to file.

Additional Inherited Members

5.25.1 Detailed Description

PDB file format.

PDB file IO subroutines for cModelHeader and cAtom array used by cModel class.

5.25.2 Member Function Documentation

5.25.2.1 void cPDBFormat::ReadFile (const char ∗const *filename,* DataHeader ∗const *header,* DataArray ∗const *data*) [virtual]

Reads array of data from file.

Parameters

filename	: file name to read.
header	: pointer to header to read in.
data	: pointer to data to read in.

Implements cDataFormat< cAtom, cModelHeader >.

5.25.2.2 void cPDBFormat::WriteFile (const char ∗const *filename,* const DataHeader ∗const *header,* const DataArray ∗const *data*) [virtual]

Writes array of data to file.

Parameters

filename	: file name to write.
header	: pointer to header to write out.
data	: pointer to data to write out.

Implements cDataFormat< cAtom, cModelHeader >.

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

- src/md_pdb_format.h
- src/md_pdb_format.cpp

5.26 cPhysAtom Struct Reference

Physical properties of an atom.

Public Attributes

- vec4_t mVelocity

  Velocity of the atom in homogeneous space (in angstroms per picosecond).
- vec4_t mAccel

  Acceleration of the atom in homogeneous space (in angstroms per sq. picosecond).
- vec_t mMassReciprocal

  Resiprocal mass (in mol/g), derived from cAtomInfo.
- vec_t mRadius

  Atomic radius (in angstroms), derived from cAtomInfo.
- vec_t mUnused [2]

  Align to 16-bytes.

5.26.1 Detailed Description

Physical properties of an atom.

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

- src/md_model.h

5.27 cPosRestraintInfo Struct Reference

Description of position restraint.

Public Attributes

- int mChain

  Index of the chain.
- int mFirstResidue

  Index of the first residue.
- int mLastResidue

  Index of the last residue.
- vec_t mHarmConst

  Harmonic constant (in $kcal/(mol*angstrom^2)$).
- bool mBackboneOnly

  True if restraint should be applied to backbone only.

5.27.1 Detailed Description

Description of position restraint.

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

- src/md_config.h

5.28 cProfileData Struct Reference

Performance profiling data.

Public Attributes

- unsigned int mECounter

  Profile energy counter.
- unsigned int mNBPLTime

  Non-bonded pair list build profile time (ms).
- unsigned int mEBondTime

  Bond energy profile time (ms).
- unsigned int mEAngleTime

  Angle energy profile time (ms).
- unsigned int mEImproperTime

  Improper energy profile time (ms).
- unsigned int mETorsionTime

  Torsion energy profile time (ms).
- unsigned int mENBTime

  Non-bonded energy profile time (ms).
- unsigned int mESpecialTime

  Special energy profile time (ms).

5.28.1 Detailed Description

Performance profiling data.

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

- src/md_model.h

5.29 cQEqParms Struct Reference

QEq parameters.

Public Attributes

- vec_t mXi

  Electronegativity.
- vec_t mJ0

  Self-repulsion.
- vec_t mQTot

  Total charge of the residue.

5.29.1 Detailed Description

QEq parameters.

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

- src/md_model.h

5.30 cResidueAtom Struct Reference

Description of a single atom of a residue.

Public Attributes

- int mIndex

    Index of the atom in the Z-matrix.
- int mPrevious [3]

    Previous atoms in the Z-matrix.
- vec_t mR

    Z-matrix R-coordinate.
- vec_t mTheta

    Z-matrix Theta-coordinate.
- vec_t mPhi

    Z-matrix Phi-coordinate.
- vec_t mCharge

    Partial charge on the atom.
- char mTitle [8]

    PDB title of the atom.
- char mFFTitle [8]

    ForceField title of the atom.
- char mSFFTitle [8]

    Solvent ForceField title of the atom.
- char mSymbol [4]

    Symbol of the atom.
- int mType

    Type of the atom (B, S, H, etc.)
- int mFlags

    Topology flags.
- int mLoopFlags

    Loop flags (bits set are loop indices)

5.30.1 Detailed Description

Description of a single atom of a residue.

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

- src/md_config.h

5.31 cResidueInfo Struct Reference

Description of a residue.

Public Attributes

- cResidueLocation mLocation [RL_MAX]

    Array of all possible locations, containing cResidueLocation records.

5.31.1 Detailed Description

Description of a residue.

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

- src/md_config.h

5.32 cResidueLocation Struct Reference

Description of a residue in a certain location.

Public Attributes

- int mNumAtoms

  Number of atoms in the residue.
- int mNumLoops

  Number of loops in the residue.
- vec_t mTotalCharge

  Total charge of the residue.
- ResidueAtomArray mAtoms

  Array of definitions of atoms, containing cResidueAtom records.

5.32.1 Detailed Description

Description of a residue in a certain location.

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

- src/md_config.h

5.33 cSemaphore Struct Reference

POSIX semaphore implementation.

Public Attributes

- pthread_mutex_t mutexBlock

  Blocking mutex for threads.
- pthread_mutex_t mutexWait

  Blocking mutex for the main thread.
- pthread_cond_t condBlock

  Conditional event for threads.
- pthread_cond_t condWait

  Conditional event for the main thread.
- int count

  Semaphore initial count.
- int maxCount

  Semaphore maximum count.

5.33.1 Detailed Description

POSIX semaphore implementation.

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

- src/md_threads.h

5.34 cSolvGSInfo Struct Reference

Description of Gaussian solvation parameters of an atom.

Public Attributes

- vec_t mVolume

  Atomic volume, in cubic angstroms.
- vec_t mGref

  Reference deltaG, in kcal/mol.
- vec_t mGfree

  Free deltaG, in kcal/mol.
- vec_t mLambda

  Correlation length, in angstroms.

5.34.1 Detailed Description

Description of Gaussian solvation parameters of an atom.

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

- src/md_config.h

5.35 cSolvParms Struct Reference

Solvation parameters.

Public Attributes

- union {
  struct {
  vec_t mR
  Van der Waals radius, in angstroms.
  vec_t mV
  Volume, in cubic angstroms.
  vec_t mA
  A = 2∗dGfree/[4∗pi∗sqrt(pi)∗lambda].
  vec_t mB
  B = -1/lambda^2.
  } **GS**
  } u

  Union for different solvent model parameters.

5.35.1 Detailed Description

Solvation parameters.

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

- src/md_model.h

5.36 cSSBond Struct Reference

Single covalent disulfide (S-S) bond of a model.

Public Attributes

- int mAtomIndices [2]

  Indices of atoms comprising the bond.
- int mResidueNumbers [2]

  Residue numbers.
- char mAtomTitles [2][8]

  Titles of atoms.

5.36.1 Detailed Description

Single covalent disulfide (S-S) bond of a model.

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

- src/md_model.h

5.37 cThreadInfo Struct Reference

Local thread information.

Public Attributes

- int mStart

  Work start.
- int mEnd

  Work end.

5.37.1 Detailed Description

Local thread information.

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

- src/md_threads.h

5.38 cThreadManager Class Reference

Thread manager object incapsulates multithreading execution capabilities.

Public Member Functions

- void Cleanup (void)

  Stop threads, free objects, etc.
- void InitThreads (void)

  Initialize threads (determine an actual thread count, create thread and sync objects, etc.).
- int CountThreads (void)

  Return number of threads in use.
- void EnterCriticalSection (void)

  Threaded function enters a critical section.
- void LeaveCriticalSection (void)

  Threaded function leaves a critical section.
- void RunThreadsOn (size_t workSize, void(∗threadWorkerFn)(int, int))

  Wrapper for global (or static class member) threaded function calls.
- void RunThreadsOn (cModel ∗pObject, size_t workSize, void(cModel::∗threadWorkerFn)(int, int))

  Wrapper for class-member threaded function calls.

Static Public Member Functions

- static void SetMaxThreadCount (int iMaxThreads)

  Sets maximum number of threads to use.
- static int MaxThreadCount (void)

  Returns maximum number of threads to use.
- static void ∗ ThreadEntryStub (void ∗pParam)

  POSIX thread entry function.

5.38.1 Detailed Description

Thread manager object incapsulates multithreading execution capabilities.

5.38.2 Member Function Documentation

5.38.2.1 void cThreadManager::SetMaxThreadCount (int *iMaxThreads*) `[static]`

Sets maximum number of threads to use.

Default is 0 (autodetect processor capabilities).

Parameters

iMaxThreads	: maximum number of threads (0 = autodetect).

5.38.2.2 static int cThreadManager::MaxThreadCount (void) `[inline]`, `[static]`

Returns maximum number of threads to use.

Returns

The maximum number of threads to use (set in SetMaxThreadCount).

5.38.2.3 void ∗ cThreadManager::ThreadEntryStub (void ∗ *pParam*) [static]

POSIX thread entry function.

Parameters

pParam	: thread parameter (thread index).

5.38.2.4 void cThreadManager::RunThreadsOn (size_t *workSize*, void(∗)(int, int) *threadWorkerFn*) [inline]

Wrapper for global (or static class member) threaded function calls.

Parameters

workSize	: number of iterations for the worker.
threadWorkerFn	: worker function.

5.38.2.5 void cThreadManager::RunThreadsOn (cModel ∗ *pObject*, size_t *workSize*, void(cModel::∗)(int, int) *threadWorkerFn*) [inline]

Wrapper for class-member threaded function calls.

Parameters

pObject	: class instance.
workSize	: number of iterations for the worker.
threadWorkerFn	: worker function (non-static member function of pObject).

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

- src/md_threads.h
- src/md_threads.cpp

5.39 cTorsion Struct Reference

Single proper torsion angle of a model.

Public Attributes

- int mAtomIndices [4]

  Indices of atoms comprising the torsion (I-J-K-L).
- bool mBarrierScale

  Boolean flag defines whether to scale barrier heights in harmonics.
- cTorsionHarm ∗ mpHarmonic

  Pointer to linked list of Fourier series describing harmonics.

5.39.1 Detailed Description

Single proper torsion angle of a model.

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

- src/md_model.h

5.40 cTorsionHarm Struct Reference

Description of a single harmonic of a torsion angle.

Public Attributes

- vec_t mHarmonicConstant

  Barrier force constant (in kcal/mol).
- vec_t mHarmonicScale

  Barrier scale (for non-ring structures).
- vec_t mPhaseCos

  Cosine of phase shift.
- vec_t mPhaseSin

  Sine of phase shift.
- int mPeriodicity

  Periodicity of a barrier.
- struct stTorsionHarm ∗ mpNext

  Pointer to the next harmonic.

5.40.1 Detailed Description

Description of a single harmonic of a torsion angle.

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

- src/md_config.h

5.41 cTorsionInfo Struct Reference

Description of a proper torsion angle.

Public Attributes

- int mFFCode [4]

  Force field codes of atoms I-J-K-L (-1 = any).
- cTorsionHarm ∗ mpHarmonic

  Fourier series representing the torsion.

5.41.1 Detailed Description

Description of a proper torsion angle.

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

- src/md_config.h

5.42 cVdWInfo Struct Reference

Description of van der Waals parameters of an atom.

Public Attributes

- vec_t mR

  Rmin/2, in angstroms.
- vec_t mE

  Well depth, in kcal/mol.

5.42.1 Detailed Description

Description of van der Waals parameters of an atom.

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

- src/md_config.h

5.43 cVdWPair Struct Reference

Van der Waals pair parameters.

Public Attributes

- vec_t mC12

  Value of C12 for the pair.
- vec_t mC6

  Value of C6 for the pair.

5.43.1 Detailed Description

Van der Waals pair parameters.

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

- src/md_model.h

5.44 FindPairByKeyFunctor< T, U > Class Template Reference

Find pair by key.

Public Member Functions

- FindPairByKeyFunctor (T check)

  Constructor.
- bool operator() (const std::pair< T, U > &check)

  Compare operator.

5.44.1 Detailed Description

template<typename T, typename U>class FindPairByKeyFunctor< T, U >

Find pair by key.

Template Parameters

T,:	first pair element type (key).
U,:	first pair element type (value).

5.44.2 Constructor & Destructor Documentation

5.44.2.1 template<typename T , typename U > FindPairByKeyFunctor< T, U >::FindPairByKeyFunctor (T *check*) [inline]

Constructor.

Parameters

check	: key to search for.

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

- src/md_common.h

5.45 FindPairByValueFunctor< T, U > Class Template Reference

Find pair by value.

Public Member Functions

- FindPairByValueFunctor (U check)

  Constructor.
- bool operator() (const std::pair< T, U > &check)

  Compare operator.

5.45.1 Detailed Description

template<typename T, typename U>class FindPairByValueFunctor< T, U >

Find pair by value.

Template Parameters

T,:	first pair element type (key).
U,:	first pair element type (value).

5.45.2 Constructor & Destructor Documentation

5.45.2.1 template<typename T , typename U > FindPairByValueFunctor< T, U >::FindPairByValueFunctor (U *check*) [inline]

Constructor.

Parameters

check	: value to search for.

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

- src/md_common.h

5.46 ResidueAtomSearchByIndexFunctor Class Reference

Residue atom search by index functor.

Public Member Functions

- ResidueAtomSearchByIndexFunctor (int index)

    Constructor.
- bool operator() (const cResidueAtom &at)

    Compare operator.

5.46.1 Detailed Description

Residue atom search by index functor.

5.46.2 Constructor & Destructor Documentation

5.46.2.1 ResidueAtomSearchByIndexFunctor::ResidueAtomSearchByIndexFunctor (int *index*) [inline]

Constructor.

Parameters

index	: index of the atom in the Z-matrix.

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

- src/md_config.h

5.47 ResidueAtomSearchByTitleFunctor Class Reference

Residue atom search by title functor.

Public Member Functions

- ResidueAtomSearchByTitleFunctor (const char ∗title)

    Constructor.
- bool operator() (const cResidueAtom &at)

Compare operator.

5.47.1 Detailed Description

Residue atom search by title functor.

5.47.2 Constructor & Destructor Documentation

5.47.2.1 ResidueAtomSearchByTitleFunctor::ResidueAtomSearchByTitleFunctor (const char $*$ *title*) `[inline]`

Constructor.

Parameters

title	: title of the atom in the Z-matrix.

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

- src/md_config.h

5.48 StringCompareFunctor Class Reference

String compare functor for IO object map.

Public Member Functions

- bool operator() (char const $*$a, char const $*$b)

  Compare operator.

5.48.1 Detailed Description

String compare functor for IO object map.

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

- src/md_common.h

5.49 TorsionInfoSearchFunctor Class Reference

Torsion info search by atomic indices.

Public Member Functions

- TorsionInfoSearchFunctor (int i, int j, int k, int l)

  Constructor.
- bool operator() (const std::list$<$ cTorsionInfo $>$::const_reference check)

  Compare operator.

5.49.1 Detailed Description

Torsion info search by atomic indices.

5.49.2 Constructor & Destructor Documentation

5.49.2.1 TorsionInfoSearchFunctor::TorsionInfoSearchFunctor (int *i*, int *j*, int *k*, int *l*) `[inline]`

Constructor.

Parameters

i	: index of atom I.
j	: index of atom J.
k	: index of atom K.
l	: index of atom L.

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

- src/md_config.h

5.50 TYPEDESCRIPTION Struct Reference

Description of parametric fields from cParameters, that are parsed.

Public Attributes

- const char ∗ mParmName

  Parameter name.
- const char ∗ mFieldName

  Field name.
- intptr_t mFieldOffset

  Field offset in the structure.
- eFieldType mFieldType

  Field type.
- bool mDefault

  Use default.

5.50.1 Detailed Description

Description of parametric fields from cParameters, that are parsed.

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

- src/md_config.h

Chapter 6

File Documentation

6.1 src/md_common.h File Reference

Common utility functions.

Classes

- class StringCompareFunctor

  String compare functor for IO object map.
- class FindPairByKeyFunctor< T, U >

  Find pair by key.
- class FindPairByValueFunctor< T, U >

  Find pair by value.
- struct cFindData

  Find data for file search funtions.

Macros

- #define offsetof(s, m) (size_t)&(((s ∗)0)->m)

  Generic "offsetof" macro.
- #define UNREFERENCED_PARAMETER(x) (void)(x)

  Generic "UNREFERENCED_PARAMETER" macro.
- #define assume_0

  "assume_0" macro (compiler optimization hint).

Typedefs

- typedef std::vector< int > IntArray

  Type for an array of integers.

Functions

- void ∗ COM_AlignedMalloc (size_t size)

  Allocate 16-byte aligned memory.
- void COM_AlignedFree (void ∗baseptr)

  Free 16-byte aligned memory.

- const char ∗ COM_FileExtension (const char ∗const name)

  Get file extension.
- void COM_Trim (char ∗string)

  Trim whitespaces from the string.
- void COM_Substr (const char ∗const string, int start, int count, char ∗buffer, size_t bufferSize)

  Extract a substring.
- char ∗ COM_AllocString (const char ∗const src)

  Returns a string allocated on heap.
- size_t COM_FileLength (FILE ∗fp)

  Returns length of a file in bytes.
- const char ∗ COM_GetHomePath (void)

  Return home path name, using xxxHOME environment variable.
- int COM_fopen_local (FILE ∗∗fp, const char ∗filename, const char ∗mode)

  Open file from the program's local directory.
- byte ∗ COM_Parse (byte ∗data, bool crossLine, char ∗token, int tokenSize, int &lineCounter)

  Parses a token out of a string.
- byte ∗ COM_MatchToken (byte ∗data, bool crossLine, const char ∗const token, int &lineCounter)

  Gets the next token and checks whether it matches the token passed.
- bool COM_TokenAvailable (byte ∗data, bool crossLine)

  Checks whether a token is available in the file data.
- int COM_Milliseconds (void)

  Return a number of milliseconds passed from program initialization.
- void COM_LogElapsedTime (void)

  Print time elapsed from program startup to log file in human-readable form.
- intptr_t COM_FindFirst (const char ∗dir, const char ∗mask, cFindData ∗pdata)

  Get the information on a matching first file, or -1 if there is no match.
- int COM_FindNext (const char ∗dir, const char ∗mask, intptr_t handle, cFindData ∗pdata)

  Get the information on the next matching file, or -1 if there is no more matches.
- int COM_FindClose (intptr_t handle)

  Close the search object.
- vec_t COM_Atof (const char ∗string)

  Helper function to avoid type cast when we use single-precision vec_t.

6.1.1 Detailed Description

Common utility functions.

6.1.2 Function Documentation

6.1.2.1 void∗ COM_AlignedMalloc (size_t *size*)

Allocate 16-byte aligned memory.

Parameters

size	: number of bytes to allocate.

6.1.2.2 void COM_AlignedFree (void ∗ *baseptr*)

Free 16-byte aligned memory.

Parameters

baseptr	: pointer to data.

6.1.2.3 const char∗ COM_FileExtension (const char ∗const *name*)

Get file extension.

Parameters

name	: file name with extension.

Returns

File extension.

6.1.2.4 void COM_Trim (char ∗ *string*)

Trim whitespaces from the string.

The function operates in place, and the original string becomes no more valid.

Parameters

string	: zero-terminated string to trim.

6.1.2.5 void COM_Substr (const char ∗const *string,* int *start,* int *count,* char ∗ *buffer,* size_t *bufferSize*)

Extract a substring.

The function operates in place, and the original string becomes no more valid.

Parameters

string	: zero-terminated source string.
start	: first character to extract.
count	: number of characters to extract.
buffer	: buffer to place the extracted substring.
bufferSize	: size of the buffer.

6.1.2.6 char∗ COM_AllocString (const char ∗const *src*)

Returns a string allocated on heap.

Parameters

src	: source string.

Returns

Pointer to the new string.

6.1.2.7 size_t COM_FileLength (FILE * *fp*)

Returns length of a file in bytes.

Parameters

fp	: file handle (FILE).

Returns

File length (in bytes).

6.1.2.8 const char* COM_GetHomePath (void)

Return home path name, using xxxHOME environment variable.

Returns

pointer to string containing home path (or default home path).

6.1.2.9 int COM_fopen_local (FILE ** *fp*, const char * *filename*, const char * *mode*)

Open file from the program's local directory.

Parameters

fp	: pointer to file handle.
filename	: file name.
mode	: file access mode ("r", "w", etc.).

Returns

error code.

6.1.2.10 byte* COM_Parse (byte * *data*, bool *crossLine*, char * *token*, int *tokenSize*, int & *lineCounter*)

Parses a token out of a string.

Parameters

data	: pointer to current file data.
crossLine	: set whether to allow reading a token from the next line.
token	: pointer to token buffer to be filled (can be NULL, to skip the token).
tokenSize	: maximum size of the token buffer.
lineCounter	: holds current line number of the parser.

Returns

pointer to the next file data to parse, NULL if nothing left unparsed.

6.1.2.11 byte* COM_MatchToken (byte * *data*, bool *crossLine*, const char *const *token*, int & *lineCounter*)

Gets the next token and checks whether it matches the token passed.

Parameters

data	: pointer to current file data.
crossLine	: set whether to allow reading a token from the next line.
token	: token to match.
lineCounter	: holds current line number of the parser.

Returns

pointer to the next file data to parse, NULL if token was not matched.

6.1.2.12 bool COM_TokenAvailable (byte ∗ *data,* bool *crossLine*)

Checks whether a token is available in the file data.

Parameters

data	: pointer to current file data.
crossLine	: set whether to allow reading a token from the next line.

Returns

true if token is available, false otherwise.

6.1.2.13 int COM_Milliseconds (void)

Return a number of milliseconds passed from program initialization.

Initialization in performed on the first call to this function.

Returns

number of milliseconds.

6.1.2.14 intptr_t COM_FindFirst (const char ∗ *dir,* const char ∗ *mask,* cFindData ∗ *pdata*)

Get the information on a matching first file, or -1 if there is no match.

Parameters

dir	: directory to search files in.
mask	: search mask.
pdata	: pointer to output file information.

Returns

search handle, -1 if no files were found.

6.1.2.15 int COM_FindNext (const char ∗ *dir,* const char ∗ *mask,* intptr_t *handle,* cFindData ∗ *pdata*)

Get the information on the next matching file, or -1 if there is no more matches.

Parameters

dir	: directory to search files in.
mask	: search mask.
handle	: search handle returned by COM_FindFirst.
pdata	: pointer to output file information.

Returns

0 (success) or -1 (fail).

6.1.2.16 int COM_FindClose (intptr_t *handle*)

Close the search object.

Parameters

handle	: search handle returned by COM_FindFirst.

Returns

0 (success) or -1 (fail).

6.2 src/md_config.h File Reference

Declaration of a config class.

Classes

- struct cAtomInfo

  Description of an atom.
- struct cResidueAtom

  Description of a single atom of a residue.
- class ResidueAtomSearchByIndexFunctor

  Residue atom search by index functor.
- class ResidueAtomSearchByTitleFunctor

  Residue atom search by title functor.
- struct cResidueLocation

  Description of a residue in a certain location.
- struct cResidueInfo

  Description of a residue.
- struct cBondInfo

  Description of a bond.
- struct cAngleInfo

  Description of a valent angle.
- struct cImproperInfo

  Description of an improper torsion angle.
- struct cTorsionHarm

  Description of a single harmonic of a torsion angle.
- struct cTorsionInfo

  Description of a proper torsion angle.

- class TorsionInfoSearchFunctor

  Torsion info search by atomic indices.
- struct cVdWInfo

  Description of van der Waals parameters of an atom.
- struct cHBInfo

  Description of H-bond parameters of an atom.
- struct cSolvGSInfo

  Description of Gaussian solvation parameters of an atom.
- struct cPosRestraintInfo

  Description of position restraint.
- struct cDistRestraintInfo

  Description of distant restraint.
- struct cParameters

  Simulation parameters that are set up via parameters' file.
- struct TYPEDESCRIPTION

  Description of parametric fields from cParameters, that are parsed.
- class cConfig

  This object incapsulates persistant molecular dynamics configuration.

Macros

- #define DEFAULT_SHORT_CUTOFF_DISTANCE 9

  Default short cut-off distance (R1), in angstroms.
- #define DEFAULT_SHORT_SWITCH_DISTANCE 7

  Default short switch distance, in angstroms.
- #define DEFAULT_LONG_CUTOFF_DISTANCE 15

  Default long cut-off distance (R2), in angstroms.
- #define DEFAULT_SS_BOND_DISTANCE vec_t(4.2)

  Default maximum length of a recognizable S-S bond.
- #define DEFAULT_SOLVENT_DIELECTRIC_CONST 80

  Default dielectric constant for a solvent (assume pure water).
- #define DEFAULT_RDIE_CONST vec_t(1.0)

  Default dielectric constant for a solvent (assume pure water).
- #define DEFAULT_SOLVENT_DEBYE_RADIUS 10000

  Default debye radius for a solvent, in angstroms (assume pure water).
- #define DEFAULT_HBOND_MODEL 128

  Default hydrogen bond model.
- #define DEFAULT_HBOND_SCALE 1

  Default hydrogen bond energy scale.
- #define DEFAULT_HBOND_CUTOFF_DISTANCE vec_t(3.5)

  Default hydrogen bond cut-off distance, in angstroms.
- #define DEFAULT_MIN_DIST_RESTR_LENGTH vec_t(0.1)

  Minimum distant restraint length, in angstroms.
- #define TF_TOPOLOGY_BEGIN (1 << 0)

  This atom is a backbone atom at the beginning of the topology.
- #define TF_TOPOLOGY_END (1 << 1)

  This atom is a backbone atom at the end of the topology.
- #define TF_TOPOLOGY_RING (1 << 2)

  This atom is a part of planar ring structure.
- #define DEFINE_PARAM_FIELD(x, y, z) { x, #y, offsetof(cParameters,y), z, true }

  Helper macro to define fields in a TYPEDESCRIPTION.

Typedefs

- typedef std::vector< cResidueAtom > ResidueAtomArray

  Type for an array of cResidueAtom objects.

Enumerations

- enum eResidueLocation {
  RL_INT = 0, **RL_BEG**, **RL_END**, **RL_ISO**,
  RL_MAX }

  Enumeration of possible residue locations.
- enum eSolvModel { **SOLV_NONE** = 0, **SOLV_GAUSS**, **SOLV_GBORN** }

  Enumeration of solvation models.
- enum eFieldType { **FIELD_INTEGER** = 0, **FIELD_FLOAT**, **FIELD_STRING**, **FIELD_FLAG** }

  Enumeration of field types used in description of parameters' fields.

Functions

- cConfig & Config (void)

  Helper function to get global config singleton.

6.2.1 Detailed Description

Declaration of a config class. The file defines the config class.

6.2.2 Function Documentation

6.2.2.1 cConfig& Config (void) [inline]

Helper function to get global config singleton.

Returns

Reference to the global config object.

6.3 src/md_format.h File Reference

Declaration of a data format interface.

Classes

- struct cDataFormat< T, U >

  Abstract data format.

6.3.1 Detailed Description

Declaration of a data format interface. The file defines data format pure abstract class.

6.4 src/md_log.h File Reference

Declaration of a log class.

Classes

- class cLog

  Object responsible for logging and error reporting.

Functions

- cLog & Log (void)

  Helper function to get global log singleton.

6.4.1 Detailed Description

Declaration of a log class. The file defines the log class.

6.4.2 Function Documentation

6.4.2.1 cLog& Log (void) [inline]

Helper function to get global log singleton.

Returns

Reference to the global log object.

6.5 src/md_main.h File Reference

Main program file, referencing all necessary includes.

Macros

- #define PROGRAM_NAME "Ambrosia"

  Program short name.
- #define PROGRAM_NAME_CAP "AMBROSIA"

  Program capitalized short name.
- #define PROGRAM_TITLE "AMBROSIA"

  Program full name.
- #define PROGRAM_BUILDSTRING __DATE__ " " __TIME__

  Program build string.
- #define PROGRAM_VERSION_MAJOR 1

  Program major version number.
- #define PROGRAM_VERSION_MINOR 0

  Program minor version number.
- #define PROGRAM_CONFIGSTRING "Release"

  Program build configuration string.
- #define DEFAULT_LOG_FILENAME PROGRAM_NAME ".log"

Program OS name.

- #define DEFAULT_MODEL_FILENAME "input.pdb"

  Default source structure file name (can be overriden with "-c" command-line argument).
- #define DEFAULT_PARMS_FILENAME "input.par"

  Default file with simulation parameters name (can be overriden with "-i" command-line argument).
- #define MAX_OSPATH 260

  Maximum size of OS file path.
- #define BIOPASED_COMPAT_PDB

  Enable some compatibilities with BioPASED.
- #define DECLARE_SINGLETON(x)

  Helper macro to declare a class as a singleton.

Functions

- void FatalExit (void)

  Exit program if fatal error has been occured.

6.5.1 Detailed Description

Main program file, referencing all necessary includes. The file defines common constants and macros, includes headers, etc.

6.5.2 Macro Definition Documentation

6.5.2.1 #define DEFAULT_LOG_FILENAME PROGRAM_NAME ".log"

Program OS name.

Default log file name (can be overriden with "-o" command-line argument).

6.5.2.2 #define DECLARE_SINGLETON(*x*)

Value:

```
\
    private: x(); \
    static x& _Instance() { static x singleton_instance; return singleton_instance; } \
    public: static x& Instance() {  typedef x& (*pfn_instance)(); static pfn_instance pf = &_Instance;
      return pf(); }
```

Helper macro to declare a class as a singleton.

6.6 src/md_math.h File Reference

Math functions.

Macros

- #define M_PI vec_t(3.14159265358979323846)

  The Pi is always the Pi.
- #define ON_EPSILON vec_t(1e-5)

  Epsilon value for convergence.

- #define GOLDEN_RATIO vec_t(1.6180339887)

  Golden ratio constant.
- #define DEG2RAD(a) (((a) ∗ M_PI) / 180)

  Macro to convert degrees to radians.
- #define RAD2DEG(a) (((a) ∗ 180) / M_PI)

  Macro to convert radians to degrees.
- #define CHECK_SIGN(a, b) (((b) > 0) ? fabs(a) : -fabs(a))

  Macro to set proper sign of a depending on b.
- #define SQR(a) ((a) ∗ (a))

  Calculate a square of a number.
- #define CUBE(a) ((a) ∗ (a) ∗ (a))

  Calculate a cube of a number.
- #define SIXTH(a) SQR((a) ∗ (a) ∗ (a))

  Calculate a sixth power of a number.
- #define EIGHTH(a) SQR((a) ∗ (a) ∗ (a) ∗ (a))

  Calculate an eighth power of a number.
- #define TWELFTH(a) SIXTH((a) ∗ (a))

  Calculate a twelfth power of a number.
- #define CLAMP(f, a, b) { if ((f) < (a)) { (f) = (a); } else if ((f) > (b)) { (f) = (b); } }

  Macro for clamping.
- #define CHECK_NAN(x)

  Debug macro for checking for a NaN value (empty in release).
- #define CHECK_NAN3(x)

  Debug macro for checking for a NaN vector (empty in release).

Functions

- bool checkNAN (float f)

  Check a single-precision IEEE-754 floating-point number for a NaN value.
- bool checkNAN (double f)

  Check a double-precision IEEE-754 floating-point number for a NaN value.
- vec_t AngleDiff (vec_t a1, vec_t a2)

  Calculate difference between angles.
- void Vec3Clear (vec_t ∗a)

  Clear a 3-component floating-point vector to zero.
- void Vec3Copy (const vec_t ∗a, vec_t ∗c)

  Copy a 3-component floating-point vector.
- void Vec3Add (const vec_t ∗a, const vec_t ∗b, vec_t ∗c)

  Add two 3-component floating-point vectors and store result.
- void Vec3Sub (const vec_t ∗a, const vec_t ∗b, vec_t ∗c)

  Subtract two 3-component floating-point vectors and store result.
- void Vec3Negate (const vec_t ∗a, vec_t ∗c)

  Negate a 3-component floating-point.
- void Vec3Scale (const vec_t ∗a, const vec_t s, vec_t ∗c)

  Scale a 3-component floating-point vector by a scalar value.
- void Vec3MA (const vec_t ∗a, const vec_t s, const vec_t ∗b, vec_t ∗c)

  Multication/addition of 3-component floating-point vectors.
- vec_t Vec3Dot (const vec_t ∗a, const vec_t ∗b)

  Dot product of two 3-component floating-point vectors.
- void Vec3Cross (const vec_t ∗a, const vec_t ∗b, vec_t ∗c)

Cross product of two 3-component floating-point vectors.
- vec_t Vec3Stp (const vec_t ∗a, const vec_t ∗b, const vec_t ∗c)

  Scalar triple product two 3-component floating-point vectors.
- vec_t Vec3Len (vec_t ∗v)

  Returns length of 3-component floating-point vector.
- vec_t Vec3LenSq (const vec_t ∗v)

  Returns square of length of 3-component floating-point vector.
- vec_t Vec3Nrm (vec_t ∗v)

  Normalizes 3-component floating-point vector in place.
- vec_t Vec3Nrm2 (vec_t ∗v, vec_t &di)

  Normalizes 3-component floating-point vector in place.
- void Vec3FastNrm (vec3_t v)

  Normalizes 3-component floating-point vector in place.
- vec_t LineMinimization (int dimension, vec4_t ∗arguments, vec4_t ∗gradient, vec_t(∗F)(vec_t))

  Line minimization of a multidimensional function.
- bool ludcmp (vec_t ∗a, const int n, int ∗indx, vec_t ∗vv)

  Replaces an n-by-n matrix, a, with the LU decomposition of a row-wise permutation of itself.
- void lubksb (const vec_t ∗a, const int n, const int ∗indx, vec_t ∗b)

  Solves the set of n linear equations Ax = b.
- void SetRandomSeed (int seed)

  Sets random seed for the generator.
- vec_t RandomFloat (vec_t flLow, vec_t flHigh)

  Generates a random float in a range [A,B].
- int RandomInt (int lLow, int lHigh)

  Generates a random integer in a range [A,B].

6.6.1 Detailed Description

Math functions.

6.6.2 Function Documentation

6.6.2.1 vec_t AngleDiff (vec_t *a1*, vec_t *a2*) [inline]

Calculate difference between angles.

Parameters

a1	: angle 1, in radians.
a2	: angle 2, in radians.

Returns

Difference between angles, in radians, modulo 2pi.

6.6.2.2 void Vec3Clear (vec_t ∗ *a*) [inline]

Clear a 3-component floating-point vector to zero.

Parameters

a	: source vector, will be cleared.

6.6.2.3 void Vec3Copy (const vec_t ∗ *a*, vec_t ∗ *c*) [inline]

Copy a 3-component floating-point vector.

Parameters

a	: source vector.
c	: destination vector.

6.6.2.4 void Vec3Add (const vec_t ∗ *a*, const vec_t ∗ *b*, vec_t ∗ *c*) [inline]

Add two 3-component floating-point vectors and store result.

Performs vector addition: C = A + B.

Parameters

a	: vector 1.
b	: vector 2.
c	: result.

6.6.2.5 void Vec3Sub (const vec_t ∗ *a*, const vec_t ∗ *b*, vec_t ∗ *c*) [inline]

Subtract two 3-component floating-point vectors and store result.

Performs vector addition: C = A - B.

Parameters

a	: menuend.
b	: subtrahend.
c	: result.

6.6.2.6 void Vec3Negate (const vec_t ∗ *a*, vec_t ∗ *c*) [inline]

Negate a 3-component floating-point.

Performs negation: C = -A.

Parameters

a	: source vector.
c	: result.

6.6.2.7 void Vec3Scale (const vec_t ∗ *a*, const vec_t *s*, vec_t ∗ *c*) [inline]

Scale a 3-component floating-point vector by a scalar value.

Performs scaling by scalar: C = A ∗ S.

Parameters

a	: source vector.
s	: scale scalar.
c	: result.

6.6.2.8 void Vec3MA (const vec_t * *a*, const vec_t *s*, const vec_t * *b*, vec_t * *c*) `[inline]`

Multication/addition of 3-component floating-point vectors.

Performs multiply-add operation: C = A + S * B.

Parameters

a	: vector 1.
s	: scale scalar.
b	: vector 2.
c	: result.

6.6.2.9 vec_t Vec3Dot (const vec_t * *a*, const vec_t * *b*) `[inline]`

Dot product of two 3-component floating-point vectors.

Calculates dot product: A . B.

Parameters

a	: vector 1.
b	: vector 2.

Returns

Scalar dot product.

6.6.2.10 void Vec3Cross (const vec_t * *a*, const vec_t * *b*, vec_t * *c*) `[inline]`

Cross product of two 3-component floating-point vectors.

Calculates cross product: C = A x B.

Parameters

a	: vector 1.
b	: vector 2.
c	: result.

6.6.2.11 vec_t Vec3Stp (const vec_t * *a*, const vec_t * *b*, const vec_t * *c*) `[inline]`

Scalar triple product two 3-component floating-point vectors.

Calculates scalar triple product: A . (B x C).

Parameters

a	: vector 1.
b	: vector 2.
c	: vector 3.

Returns

Scalar triple product.

6.6.2.12 vec_t Vec3Len (vec_t ∗ *v*) [inline]

Returns length of 3-component floating-point vector.

Parameters

v	: source vector.

Returns

Length of the source vector.

6.6.2.13 vec_t Vec3LenSq (const vec_t ∗ *v*) [inline]

Returns square of length of 3-component floating-point vector.

Parameters

v	: source vector.

Returns

Square of length of the source vector.

6.6.2.14 vec_t Vec3Nrm (vec_t ∗ *v*) [inline]

Normalizes 3-component floating-point vector in place.

Parameters

v	: vector to normalize, will be overwritten with result.

Returns

Length of the original (non-normalized) vector.

6.6.2.15 vec_t Vec3Nrm2 (vec_t ∗ *v*, vec_t & *di*) [inline]

Normalizes 3-component floating-point vector in place.

Parameters

v	: vector to normalize, will be overwritten with result.
di	: inverse of the length of the original (non-normalized) vector.

Returns

Length of the original (non-normalized) vector.

6.6.2.16 void Vec3FastNrm (vec3_t *v*) [inline]

Normalizes 3-component floating-point vector in place.

This can a bit faster than Vec3Nrm, since no checks are performed and nothing is returned.

Parameters

v	: vector to normalize, will be overwritten with result.

6.6.2.17 vec_t LineMinimization (int *dimension*, vec4_t ∗ *arguments*, vec4_t ∗ *gradient*, vec_t(∗)(vec_t) *F*)

Line minimization of a multidimensional function.

This is used in conjugate gradients energy optimization algorithm.

Parameters

dimension	: number of arguments of a multidimensional function.
arguments	: array of arguments (will be modified).
gradient	: array of derivatives (will be modified).
F	: interface to multidimensional function.

Returns

Value of a function at the minimum.

Adapted function linmin from Numeric Recipes.

6.6.2.18 bool ludcmp (vec_t ∗ *a*, const int *n*, int ∗ *indx*, vec_t ∗ *vv*)

Replaces an n-by-n matrix, a, with the LU decomposition of a row-wise permutation of itself.

Parameters

a	: source matrix.
n	: dimension of the matrix (matrix is square).
indx	: the vector which records the row permutation effected by the partial pivoting.
vv	: temporary matrix (dimension is the same as for a).

Returns

True if LU decomposition is successful, false if there is a singularity.

LUDCMP Replaces an n-by-n matrix, a, with the LU decomposition of a row-wise permutation of itself

6.6.2.19 void lubksb (const vec_t ∗ *a*, const int *n*, const int ∗ *indx*, vec_t ∗ *b*)

Solves the set of n linear equations Ax = b.

LUBKSB must be used with the procedure LUDCMP to do this.

Parameters

a	: LU matrix.
n	: dimension of the LU matrix (matrix is square).
indx	: the vector which holds the row permutation effected by the partial pivoting (from LUDCMP).
b	: input values of b; output results.

LUBKSB Solves the set of n linear equations Ax = b (LUBKSB must be used with the procedure LUDCMP to do this)

6.6.2.20 void SetRandomSeed (int *seed*)

Sets random seed for the generator.

Parameters

seed	: seed value (0 = use current time).

6.6.2.21 vec_t RandomFloat (vec_t *flLow*, vec_t *flHigh*)

Generates a random float in a range [A,B].

Parameters

flLow	: minimum allowed value.
flHigh	: maximum allowed value.

Returns

A random float value in a range.

6.6.2.22 int RandomInt (int *lLow*, int *lHigh*)

Generates a random integer in a range [A,B].

Parameters

lLow	: minimum allowed value.
lHigh	: maximum allowed value.

Returns

A random integer value in a range.

6.7 src/md_model.h File Reference

Declaration of a model class.

Classes

- struct cModelHeader

  Model header (global data).
- struct cConnectivity

  Connectivity information for an atom.
- struct cVdWPair

  Van der Waals pair parameters.
- struct cHBPair

  Hydrogen bonding pair parameters.
- struct cSolvParms

  Solvation parameters.
- struct cQEqParms

  QEq parameters.

- struct cAtom

  Single atom of a model.
- struct cPhysAtom

  Physical properties of an atom.
- struct cBond

  Single covalent bond of a model.
- struct cSSBond

  Single covalent disulfide (S-S) bond of a model.
- struct cDistRestrain

  Single distant restraint of a model.
- struct cAngle

  Single covalent angle of a model.
- struct cImproper

  Single improper torsion angle of a model.
- struct cTorsion

  Single proper torsion angle of a model.
- struct cNonBondedPair

  Non-bonded pair.
- struct cHBTriplet

  Hydrogen bond triplet (D = donor, H = hydrogen, A = acceptor).
- class AtomSearchFunctor

  Atom search functor.
- class AtomSortFunctor

  Atom sort functor.
- struct cProfileData

  Performance profiling data.
- class cModel

  Object representing a 3D-model of biopolymer.

Macros

- #define MAX_PROFILER_DEPTH 8

  Maximum depth of nested functions to profile.
- #define AF_HEAVY (1 << 0)

  Atom is heavy (i.e. not hydrogen).
- #define AF_HB_DONOR (1 << 1)

  This atom is a potential hydrogen bond donor.
- #define AF_HB_ACCEPTOR (1 << 2)

  This atom is a potential hydrogen bond acceptor.
- #define AF_RL_BEGIN (1 << 3)

  Residue is at the beginning of a chain.
- #define AF_RL_END (1 << 4)

  Residue is at the end of a chain.
- #define AF_RESTRAINED (1 << 5)

  Atom's position is harmonically restrained.
- #define NBPF_NEIGHBOURS_14 (1 << 0)

  Non-bonded pair flag: 1-4 connection.
- #define NBPF_BOTH_HEAVY (1 << 1)

  Non-bonded pair flag: both atoms are heavy.

Functions

- cModel & Model (void)

  Helper function to get global model singleton.

6.7.1 Detailed Description

Declaration of a model class. The file defines the model class.

6.7.2 Function Documentation

6.7.2.1 cModel& Model (void) [inline]

Helper function to get global model singleton.

Returns

Reference to the global model object.

6.8 src/md_pdb_format.h File Reference

Declaration of a PDB file IO class.

Classes

- class cPDBFormat

  PDB file format.

6.8.1 Detailed Description

Declaration of a PDB file IO class. The file defines PDB data format IO operations.

6.9 src/md_scrti.h File Reference

Secure CRT implementation.

Macros

- #define sprintf_s(buffer, buffer_size, stringbuffer,...) sprintf(buffer, stringbuffer, __VA_ARGS__)

  Macro replacement for sprintf_s.
- #define _vsnprintf_s(buffer, buffer_size, n, format, arg) vsnprintf(buffer, n, format, arg)

  Macro replacement for _vsnprintf_s.
- #define _stricmp strcasecmp

  Macro replacement for _stricmp.
- #define _strnicmp strncasecmp

  Macro replacement for _strnicmp.

Functions

- int fopen_s (FILE ∗∗f, const char ∗name, const char ∗mode)

  Implementation of fopen_s.
- int strcpy_s (char ∗strDestination, size_t numberOfElements, const char ∗strSource)

  Implementation of strcpy_s.
- int strcpy_s (char ∗strDestination, const char ∗strSource)

  Implementation of strcpy_s.
- int strncpy_s (char ∗strDestination, size_t numberOfElements, const char ∗strSource, size_t count)

  Implementation of strncpy_s.
- int strncpy_s (char ∗strDestination, const char ∗strSource, size_t count)

  Implementation of strncpy_s.
- int strcat_s (char ∗strDestination, size_t numberOfElements, const char ∗strSource)

  Implementation of strcat_s.
- int strcat_s (char ∗strDestination, const char ∗strSource)

  Implementation of strcat_s.
- int strncat_s (char ∗strDestination, size_t numberOfElements, const char ∗strSource, size_t count)

  Implementation of strncat_s.
- int strncat_s (char ∗strDestination, const char ∗strSource, size_t count)

  Implementation of strncat_s.
- int _strupr_s (char ∗str)

  Implementation of _strupr_s.
- int _strupr_s (char ∗str, size_t numberOfElements)

  Implementation of _strupr_s.
- int getenv_s (size_t ∗pReturnValue, char ∗buffer, size_t numberOfElements, const char ∗varname)

  Implementation of getenv_s.

6.9.1 Detailed Description

Secure CRT implementation. The file declares secure CRT function prototypes with a custom implementation for compilers that don't support them (e.g. GCC).

6.10 src/md_threads.h File Reference

Declaration of a thread class.

Classes

- struct cThreadInfo

  Local thread information.
- struct cSemaphore

  POSIX semaphore implementation.
- class cThreadManager

  Thread manager object incapsulates multithreading execution capabilities.

Macros

- #define MAXTHREADS 64

  Maximum number of threads supported.

Typedefs

- typedef void(∗ cThreadFunc)(int, int)

  Define this macro to ignore multithreading capabilities (everything will be executed in the main thread).
- typedef void(cModel::∗ cModelThreadFunc)(int, int)

  Thread class-member function pointer.

Functions

- cThreadManager & ThreadManager (void)

  Helper function to get global thread manager singleton.

6.10.1 Detailed Description

Declaration of a thread class. The file defines the thread class.

6.10.2 Typedef Documentation

6.10.2.1 typedef void(∗ cThreadFunc)(int, int)

Define this macro to ignore multithreading capabilities (everything will be executed in the main thread).

Define this macro to debug multithreading. Thread function pointer.

6.10.3 Function Documentation

6.10.3.1 cThreadManager& ThreadManager (void) [inline]

Helper function to get global thread manager singleton.

Returns

Reference to the global thread manager object.

6.11 src/md_types.h File Reference

Custom data type definitions.

Typedefs

- typedef unsigned char byte

  Unsigned 8-bit integer.
- typedef unsigned short word

  Unsigned 16-bit integer.
- typedef unsigned int dword

  Unsigned 32-bit integer.
- typedef unsigned long long qword

  Unsigned 64-bit integer.
- typedef double vec_t

  Double-precision floating-point value.
- typedef vec_t vec2_t [2]

2-component floating-point vector.
- typedef vec_t vec3_t [3]

  3-component floating-point vector.
- typedef vec_t vec4_t [4]

  4-component floating-point vector.

6.11.1 Detailed Description

Custom data type definitions. The file defines custom data types.

6.11.2 Typedef Documentation

6.11.2.1 typedef double vec_t

Double-precision floating-point value.

May be changed to single (float), if such precision is a must, or you are RAM-limited (NOT recommended!).

Index