InterMol Documentation

Release 0.2a1

Christoph Klein, Christopher Lee, Ellen Zhong, and Michael Shirts

January 25, 2015

Contents

CHAPTER 1

Installation

1.1 Installation

1.1.1 Install with pip (coming soon!)

InterMol will be added to PyPI as soon as we get our first stable release up and running.

1.1.2 Install from source

```
$ git clone https://github.com/shirtsgroup/InterMol.git
$ cd InterMol
$ python setup.py install
```

Or if you plan on contributing something:

```
$ python setup.py develop
```

1.1.3 Dependencies

To use InterMol, the following libraries and software will need to be installed.

Linux, Mac OS X or Windows operating system We develop mainly on 64-bit Mac and CentOS machines. TravisCI is currently only set up to perform testing on Debian.

Python == 2.7 Once our unit tests flesh out a bit more, we intend to add support for >=2.6.

NumPy >= 1.6.0 Numpy is the base package for numerical computing in python.

simtk.unit >= 0.1 Python unit classes for dimensional analysis and unit conversion.

CHAPTER 2

Development

2.1 Contributing

Contributions are welcome and they are greatly appreciated! Every little bit helps and credit will always be given.

2.1.1 Bug Reports

If you find a bug, please file a detailed issue here and **provide input files** so that we can try to reproduce the problem.

2.1.2 Code Style

PEP8 describes the style guide for python code. Please try to follow it when reasonable, in particular the naming conventions (e.g. public methods should use `lower_case_with_underscores` instead of `camelCase`).

Two helpful tools for checking your code are `flake8` and `pylint`. The latter is a lot pickier but generally, your code should pass most `flake8` tests:

```
$ pip install flake8 pylint
```

Also some IDE's like PyCharm will notify you of PEP8 violations as you code.

Note: Some older chunks of the code do not yet adhere to PEP8. If you find something easily fixable, please do so!

2.1.3 Running our tests

InterMol uses py.test for unit testing. To run them simply type run the following while in the base directory:

```
$ py.test
```

We need a LOT more tests so any help here is especially welcome!

To debug failing tests, you typically get a clearer output by running a subset of the tests, e.g.:

```
$ python test_gromacs.py --type unit
```

which prints out log files to `intermol/tests/unit_test_outputs/from_gromacs/[system name]/debug.log`. Re-running a single test is best done directly via the *convert.py* script, e.g.:

```
$ python convert.py --gro_in tests/gromacs/unit_tests/[system name]/[system name].{top,gro} --gromacs
```

Note: If you have any ideas or suggestions for streamlining the testing process please let us know by filing an issue or opening a pull request!

2.1.4 Git Flow

Because we are supporting multiple molecular dynamics engines that should all work independently, we try to keep development of each engine in a separate branch until the basics are working.

To this end, we've started working with the git flow branching model. The basic things to know are:

1. The `master` branch is strictly used for releases.
2. The `develop` branch is where (!!!) development happens.
3. When we start working on a new engine, we create a feature branch. E.g., at the time of this writing, there are branches called `feature/lammps` and `feature/desmond`. Once the overall structure in this branch is fairly stable and has a good amounts of tests, we merge it into `develop`.

So what do you, the interested developer, need to know?

1. Don't make pull requests against `master`.
2. Choose either `develop` or the appropriate `feature/*` branch to pull against.

For more reading and a neat tool to help with branching see:

http://jeffkreeftmeijer.com/2010/why-arent-you-using-git-flow/

https://github.com/nvie/gitflow

http://danielkummer.github.io/git-flow-cheatsheet/

CHAPTER 3

API Reference

3.1 intermol package

3.1.1 Subpackages

intermol.forces package

Submodules

intermol.forces.abstract_2_virtual_type module

class intermol.forces.abstract_2_virtual_type.**Abstract2VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

placeholder

intermol.forces.abstract_3_virtual_type module

class intermol.forces.abstract_3_virtual_type.**Abstract3VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

placeholder

intermol.forces.abstract_4_virtual_type module

class intermol.forces.abstract_4_virtual_type.**Abstract4VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*, *bondingtype5*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

intermol.forces.abstract_angle_type module

class intermol.forces.abstract_angle_type.**AbstractAngleType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *c=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

c

intermol.forces.abstract_atom_type module

class intermol.forces.abstract_atom_type.**AbstractAtomType**(*atomtype*, *bondtype=None*, *atomic_number=None*, *mass=None*, *charge=None*, *ptype=None*)

Bases: intermol.forces.abstract_type.AbstractType

atomic_number

atomtype

bondtype

charge

mass

ptype

intermol.forces.abstract_bond_type module

class intermol.forces.abstract_bond_type.**AbstractBondType**(*bondingtype1*, *bondingtype2*, *order=1*, *c=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

c

order

intermol.forces.abstract_dihedral_type module

class intermol.forces.abstract_dihedral_type.**AbstractDihedralType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*, *improper=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

improper

intermol.forces.abstract_nonbonded_type module

class intermol.forces.abstract_nonbonded_type.**AbstractNonbondedType**(*atom1*, *atom2*, *type*)

Bases: intermol.forces.abstract_type.AbstractType

atom1

atom2

type

intermol.forces.abstract_pair_type module

class intermol.forces.abstract_pair_type.**AbstractPairType**(*bondingtype1*, *bondingtype2*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

long

scaleLJ

scaleQQ

intermol.forces.abstract_type module

class intermol.forces.abstract_type.**AbstractType**

Bases: object

__repr__()

Print the object and all of its non-magic attributes.

intermol.forces.atom_c_type module

class intermol.forces.atom_c_type.**AtomCType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_atom_type.AbstractAtomType

intermol.forces.atom_sigeps_type module

class intermol.forces.atom_sigeps_type.**AtomSigepsType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_atom_type.AbstractAtomType

intermol.forces.buckingham_nonbonded_type module

class intermol.forces.buckingham_nonbonded_type.**BuckinghamNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *a=Quantity(value=0.0, unit=kilojoule/mole)*, *b=Quantity(value=0.0, unit=/nanometer)*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *type=False*)

Bases: intermol.forces.buckingham_nonbonded_type.BuckinghamNonbondedType

stub documentation

class intermol.forces.buckingham_nonbonded_type.**BuckinghamNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

C6

a

b

type

intermol.forces.connection_bond_type module

class intermol.forces.connection_bond_type.**ConnectionBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *bondingorder=1*, *c=False*)

Bases: intermol.forces.connection_bond_type.ConnectionBondType

stub documentation

class intermol.forces.connection_bond_type.**ConnectionBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

order

intermol.forces.convert_dihedrals module

intermol.forces.convert_dihedrals.**convert_dihedral_from_OPLS_to_RB**(*f*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_RB_to_OPLS**(*c*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_RB_to_trig**(*c*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_fourier_to_trig**(*f*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_proper_to_trig**(*p*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_RB**(*fcs*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_fourier**(*fcs*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_proper**(*fcs*, *convention='0'*)

intermol.forces.convert_dihedrals.**convert_nothing**(*x*)
useful utility for not converting anything

intermol.forces.cosine_angle_type module

class intermol.forces.cosine_angle_type.**CosineAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *c=False*)
Bases: intermol.forces.cosine_angle_type.CosineAngleType

http://lammps.sandia.gov/doc/angle_cosine.html

class intermol.forces.cosine_angle_type.**CosineAngleType**(**args*, ***kwds*)
Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

intermol.forces.cosine_squared_angle_type module

class intermol.forces.cosine_squared_angle_type.**CosineSquaredAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *c=False*)
Bases: intermol.forces.cosine_squared_angle_type.CosineSquaredAngleType

stub documentation

class intermol.forces.cosine_squared_angle_type.**CosineSquaredAngleType**(**args*, ***kwds*)
Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

theta

intermol.forces.cross_bond_angle_angle_type module

class intermol.forces.cross_bond_angle_angle_type.**CrossBondAngleAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *r1=Quantity(value=0.0, unit=nanometer)*, *r2=Quantity(value=0.0, unit=nanometer)*, *r3=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mo*, *c=False*)

Bases: intermol.forces.cross_bond_angle_angle_type.CrossBondAngleAngleType

stub documentation

class intermol.forces.cross_bond_angle_angle_type.**CrossBondAngleAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

r1

r2

r3

intermol.forces.cross_bond_bond_angle_type module

class intermol.forces.cross_bond_bond_angle_type.**CrossBondBondAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *r1=Quantity(value=0.0, unit=nanometer)*, *r2=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole)*, *c=False*)

Bases: intermol.forces.cross_bond_bond_angle_type.CrossBondBondAngleType

stub documentation

class intermol.forces.cross_bond_bond_angle_type.**CrossBondBondAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

r1

r2

intermol.forces.cubic_bond_type module

class intermol.forces.cubic_bond_type.**CubicBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *C2=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *C3=Quantity(value=0.0, unit=kilojoule/(nanometer**3*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.cubic_bond_type.CubicBondType

stub documentation

class intermol.forces.cubic_bond_type.**CubicBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

C2

C3

c

length

order

intermol.forces.fene_bond_type module

class intermol.forces.fene_bond_type.**FeneBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *kb=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.fene_bond_type.FeneBondType

stub documentation

class intermol.forces.fene_bond_type.**FeneBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

kb

length

order

intermol.forces.fene_expandable_bond_type module

class intermol.forces.fene_expandable_bond_type.**FeneExpandableBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *length=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *sigma=Quantity(value=0.0, unit=nanometer)*, *delta=Quantity(value=0.0, unit=nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.fene_expandable_bond_type.FeneExpandableBondType

stub documentation

class intermol.forces.fene_expandable_bond_type.**FeneExpandableBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

delta

epsilon

k

length

order

sigma

intermol.forces.forcedata module

intermol.forces.forcefunctions module

intermol.forces.forcefunctions.**build_paramlist**(*program*)

Create a paramlist specific for a given program.

intermol.forces.forcefunctions.**build_unitvars**(*program*, *paramlist*, *dumself=None*)

Takes a string program name (one of the supported programs), and a 'self' object it looks like the keyword is not being used, but it is used in the line eval(unit). The test name 'dumself' needs to match what is in the force data arrays. Currently only used for lammps.

intermol.forces.forcefunctions.**capifyname**(*forcename*)

Return name of the class in camelCase.

intermol.forces.forcefunctions.**create_kwd_dict**(*unitvars*, *paramlist*, *force_type_object*, *values*, *optvalues=None*)

intermol.forces.forcefunctions.**create_kwds_from_entries**(*unitvars*, *paramlist*, *entries*, *force_type*, *offset=0*)

Create a keyword dictionary given an array of information from a file format

requires the master set of units, the master set of parameter lists, an object (either a force_class or force_type), the list of information to be converted into a keyword, and an offset.

Parameters **offset** (*int*) – how far over from the first entry we translate

intermol.forces.forcefunctions.**get_parameter_kwds_from_force**(*force*, *forceparams*, *paramlist*)

intermol.forces.forcefunctions.**get_parameter_list_from_force**(*force*, *paramlist*)

Create a function that returns the paramters of a function type.

First, we need make some additions to the parameter list dictionary, which we do once when the forcedata script is imported. Useful to put the forces here as well. We won't make this a function for now since it's needed in this module.

intermol.forces.forcefunctions.**get_parameter_list_from_kwds**(*force*, *kwds*, *paramlist*)

intermol.forces.forcefunctions.**optforceparams**(*force_type*, *forcetype_object=None*)

Return the dictionary of optional paramters of an abstract force type.

If no object is given, we fill with blanks.

intermol.forces.forcefunctions.**optparamkeylookup**(*force_type*)

Given a force_type object, determine the key associated with the optional parameters.

intermol.forces.forcefunctions.**optparamlookup**(*force_type_object*, *object_default=False*)

A wrapper for optforceparams that takes a force_type object and returns the optional parameter dictionary.

intermol.forces.forcefunctions.**specify**(*program_units*, *unitset*, *dumself=None*, *shouldEval=True*)

Takes the dict of units, and a set of dimensions and replaces the dimensions with the appropriate units.

intermol.forces.four_fdn_virtual_type module

class intermol.forces.four_fdn_virtual_type.**FourFdnVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *atom5*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *bondingtype5=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *c=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.four_fdn_virtual_type.FourFdnVirtualType

stub documentation

class intermol.forces.four_fdn_virtual_type.**FourFdnVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_4_virtual_type.Abstract4VirtualType

a

b

c

placeholder

intermol.forces.fourier_dihedral_type module

class intermol.forces.fourier_dihedral_type.**FourierDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *c1=Quantity(value=0.0, unit=kilojoule/mole)*, *c2=Quantity(value=0.0, unit=kilojoule/mole)*, *c3=Quantity(value=0.0, unit=kilojoule/mole)*, *c4=Quantity(value=0.0, unit=kilojoule/mole)*, *c5=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.fourier_dihedral_type.FourierDihedralType

stub documentation

class intermol.forces.fourier_dihedral_type.**FourierDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

c1

c2

c3

c4

c5

improper

intermol.forces.g96_bond_type module

class intermol.forces.g96_bond_type.**G96Bond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**4*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.g96_bond_type.G96BondType

stub documentation

class intermol.forces.g96_bond_type.**G96BondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.harmonic_angle_type module

class intermol.forces.harmonic_angle_type.**HarmonicAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *c=False*)

Bases: intermol.forces.harmonic_angle_type.HarmonicAngleType

stub documentation

class intermol.forces.harmonic_angle_type.**HarmonicAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

theta

intermol.forces.harmonic_bond_type module

class intermol.forces.harmonic_bond_type.**HarmonicBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.harmonic_bond_type.HarmonicBondType

stub documentation

class intermol.forces.harmonic_bond_type.**HarmonicBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.harmonic_potential_bond_type module

class intermol.forces.harmonic_potential_bond_type.**HarmonicPotentialBond**(*atom1*, *atom2*, *bonding-type1=None*, *bonding-type2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*, *order=1*, *c=False*)

Bases: intermol.forces.harmonic_potential_bond_type.HarmonicPotentialBondType

stub documentation

class intermol.forces.harmonic_potential_bond_type.**HarmonicPotentialBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.improper_harmonic_dihedral_type module

class intermol.forces.improper_harmonic_dihedral_type.**ImproperHarmonicDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bonding-type1=None*, *bonding-type2=None*, *bonding-type3=None*, *bonding-type4=None*, *xi=Quantity(value=0*, *unit=degree)*, *k=Quantity(value=0*, *unit=kilojoule/(mole*, *improper=False*)

Bases: intermol.forces.improper_harmonic_dihedral_type.ImproperHarmonicDihedralType

stub documentation

class intermol.forces.improper_harmonic_dihedral_type.**ImproperHarmonicDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

improper

k

xi

intermol.forces.lj_c_nonbonded_type module

class intermol.forces.lj_c_nonbonded_type.**LjCNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *type=False*)

Bases: intermol.forces.lj_c_nonbonded_type.LjCNonbondedType

stub documentation

class intermol.forces.lj_c_nonbonded_type.**LjCNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

C12

C6

type

intermol.forces.lj_c_pair_type module

class intermol.forces.lj_c_pair_type.**LjCPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_c_pair_type.LjCPairType

stub documentation

class intermol.forces.lj_c_pair_type.**LjCPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

C12

C6

long

scaleLJ

scaleQQ

intermol.forces.lj_default_pair_type module

class intermol.forces.lj_default_pair_type.**LjDefaultPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_default_pair_type.LjDefaultPairType

stub documentation

class intermol.forces.lj_default_pair_type.**LjDefaultPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

long

scaleLJ

scaleQQ

intermol.forces.lj_sigeps_nonbonded_type module

class intermol.forces.lj_sigeps_nonbonded_type.**LjSigepsNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *type=False*)

Bases: intermol.forces.lj_sigeps_nonbonded_type.LjSigepsNonbondedType

stub documentation

class intermol.forces.lj_sigeps_nonbonded_type.**LjSigepsNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

epsilon

sigma

type

intermol.forces.lj_sigeps_pair_type module

class intermol.forces.lj_sigeps_pair_type.**LjSigepsPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_sigeps_pair_type.LjSigepsPairType

stub documentation

class intermol.forces.lj_sigeps_pair_type.**LjSigepsPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

epsilon

long

scaleLJ

scaleQQ

sigma

intermol.forces.ljq_c_pair_type module

class intermol.forces.ljq_c_pair_type.**LjqCPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *qi=Quantity(value=0.0, unit=elementary charge)*, *qj=Quantity(value=0.0, unit=elementary charge)*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_c_pair_type.LjqCPairType

stub documentation

class intermol.forces.ljq_c_pair_type.**LjqCPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

C12

C6

long

qi

qj

scaleLJ

scaleQQ

intermol.forces.ljq_default_pair_type module

class intermol.forces.ljq_default_pair_type.**LjqDefaultPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_default_pair_type.LjqDefaultPairType

stub documentation

class intermol.forces.ljq_default_pair_type.**LjqDefaultPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

long

scaleLJ

scaleQQ

intermol.forces.ljq_sigeps_pair_type module

class intermol.forces.ljq_sigeps_pair_type.**LjqSigepsPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *qi=Quantity(value=0.0, unit=elementary charge)*, *qj=Quantity(value=0.0, unit=elementary charge)*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_sigeps_pair_type.LjqSigepsPairType

stub documentation

class intermol.forces.ljq_sigeps_pair_type.**LjqSigepsPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

epsilon

long

qi

qj

scaleLJ

scaleQQ

sigma

intermol.forces.make_forces module

intermol.forces.morse_bond_type module

class intermol.forces.morse_bond_type.**MorseBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *D=Quantity(value=0.0, unit=kilojoule/mole)*, *beta=Quantity(value=0.0, unit=/nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.morse_bond_type.MorseBondType

stub documentation

class intermol.forces.morse_bond_type.**MorseBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

D

beta

c

length

order

intermol.forces.nonlinear_bond_type module

class intermol.forces.nonlinear_bond_type.**NonlinearBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *r0=Quantity(value=0.0, unit=nanometer)*, *lamda=Quantity(value=0.0, unit=nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.nonlinear_bond_type.NonlinearBondType

http://lammps.sandia.gov/doc/bond_nonlinear.html

class intermol.forces.nonlinear_bond_type.**NonlinearBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

epsilon

lamda

order

r0

intermol.forces.proper_periodic_dihedral_type module

class intermol.forces.proper_periodic_dihedral_type.**ProperPeriodicDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *phi=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *multiplicity=Quantity(value=0.0, unit=dimensionless)*, *weight=Quantity(value=0.0, unit=dimensionless)*, *improper=False*)

Bases: intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedralType

stub documentation

class intermol.forces.proper_periodic_dihedral_type.**ProperPeriodicDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

improper

k

multiplicity

phi

weight

intermol.forces.quartic_angle_type module

class intermol.forces.quartic_angle_type.**QuarticAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *C0=Quantity(value=0.0, unit=kilojoule/mole)*, *C1=Quantity(value=0.0, unit=kilojoule/(mole*radian))*, *C2=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *C3=Quantity(value=0.0, unit=kilojoule/(mole*radian**3))*, *C4=Quantity(value=0.0, unit=kilojoule/(mole*radian**4))*, *c=False*)

Bases: intermol.forces.quartic_angle_type.QuarticAngleType

stub documentation

class intermol.forces.quartic_angle_type.**QuarticAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

C0

C1

C2

C3

C4

c

theta

intermol.forces.quartic_bond_type module

class intermol.forces.quartic_bond_type.**QuarticBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *C2=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *C3=Quantity(value=0.0, unit=kilojoule/(nanometer**3*mole))*, *C4=Quantity(value=0.0, unit=kilojoule/(nanometer**4*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.quartic_bond_type.QuarticBondType

stub documentation

class intermol.forces.quartic_bond_type.**QuarticBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

C2

C3

C4

c

length

order

intermol.forces.quartic_breakable_bond_type module

class intermol.forces.quartic_breakable_bond_type.**QuarticBreakableBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**4*m*, *B1=Quantity(value=0.0, unit=nanometer)*, *B2=Quantity(value=0.0, unit=nanometer)*, *Rc=Quantity(value=0.0, unit=nanometer)*, *U0=Quantity(value=0.0, unit=kilojoule/mole)*, *order=1*, *c=False*)

Bases: intermol.forces.quartic_breakable_bond_type.QuarticBreakableBondType

http://lammps.sandia.gov/doc/bond_quartic.html

class intermol.forces.quartic_breakable_bond_type.**QuarticBreakableBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

B1

B2

Rc

U0

c

k

order

intermol.forces.rb_dihedral_type module

class intermol.forces.rb_dihedral_type.**RbDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *C0=Quantity(value=0.0, unit=kilojoule/mole)*, *C1=Quantity(value=0.0, unit=kilojoule/mole)*, *C2=Quantity(value=0.0, unit=kilojoule/mole)*, *C3=Quantity(value=0.0, unit=kilojoule/mole)*, *C4=Quantity(value=0.0, unit=kilojoule/mole)*, *C5=Quantity(value=0.0, unit=kilojoule/mole)*, *C6=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.rb_dihedral_type.RbDihedralType

stub documentation

class intermol.forces.rb_dihedral_type.**RbDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

C0

C1

C2

C3

C4

C5

C6

improper

intermol.forces.settles module

class intermol.forces.settles.**Settles**(**args*, ***kwds*)

Bases: intermol.forces.abstract_type.AbstractType

intermol.forces.three_fad_virtual_type module

class intermol.forces.three_fad_virtual_type.**ThreeFadVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *theta=Quantity(value=0.0, unit=degree)*, *d=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_fad_virtual_type.ThreeFadVirtualType

stub documentation

class intermol.forces.three_fad_virtual_type.**ThreeFadVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

d

placeholder

theta

intermol.forces.three_fd_virtual_type module

class intermol.forces.three_fd_virtual_type.**ThreeFdVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *d=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_fd_virtual_type.ThreeFdVirtualType

stub documentation

class intermol.forces.three_fd_virtual_type.**ThreeFdVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

d

placeholder

intermol.forces.three_linear_virtual_type module

class intermol.forces.three_linear_virtual_type.**ThreeLinearVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *placeholder=False*)

Bases: intermol.forces.three_linear_virtual_type.ThreeLinearVirtualType

stub documentation

class intermol.forces.three_linear_virtual_type.**ThreeLinearVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

b

placeholder

intermol.forces.three_out_virtual_type module

class intermol.forces.three_out_virtual_type.**ThreeOutVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *c=Quantity(value=0.0, unit=/nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_out_virtual_type.ThreeOutVirtualType

stub documentation

class intermol.forces.three_out_virtual_type.**ThreeOutVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

b

c

placeholder

intermol.forces.trig_dihedral_type module

class intermol.forces.trig_dihedral_type.**TrigDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *phi=Quantity(value=0.0, unit=degree)*, *fc0=Quantity(value=0.0, unit=kilojoule/mole)*, *fc1=Quantity(value=0.0, unit=kilojoule/mole)*, *fc2=Quantity(value=0.0, unit=kilojoule/mole)*, *fc3=Quantity(value=0.0, unit=kilojoule/mole)*, *fc4=Quantity(value=0.0, unit=kilojoule/mole)*, *fc5=Quantity(value=0.0, unit=kilojoule/mole)*, *fc6=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.trig_dihedral_type.TrigDihedralType

stub documentation

class intermol.forces.trig_dihedral_type.**TrigDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

fc0

fc1

fc2

fc3

fc4

fc5

fc6

improper

phi

intermol.forces.two_virtual_type module

class intermol.forces.two_virtual_type.**TwoVirtual**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *placeholder=False*)

Bases: intermol.forces.two_virtual_type.TwoVirtualType

stub documentation

class intermol.forces.two_virtual_type.**TwoVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_2_virtual_type.Abstract2VirtualType

a

placeholder

intermol.forces.urey_bradley_angle_type module

class intermol.forces.urey_bradley_angle_type.**UreyBradleyAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *r=Quantity(value=0.0, unit=nanometer)*, *kUB=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *c=False*)

Bases: intermol.forces.urey_bradley_angle_type.UreyBradleyAngleType

stub documentation

class intermol.forces.urey_bradley_angle_type.**UreyBradleyAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

kUB

r

theta

Module contents

intermol.gromacs package

Submodules

intermol.gromacs.grofile_parser module

class intermol.gromacs.grofile_parser.**GromacsGroParser**(*gro_file*)

Bases: object

GromacsGroParser reads and writes Gromacs .gro files

A .gro file also contains some topological information, such as elements and residue names, but not enough to construct a full Topology object. This information is recorded and stored in the object's public fields.

read()

write(*system*)

Write the system out in a Gromacs 4.6 format

Parameters **filename** (*str*) – the file to write out to

intermol.gromacs.gromacs_driver module

intermol.gromacs.gromacs_driver.**gromacs_energies**(*top=None*, *gro=None*, *mdp=None*, *gropath=None*, *grosuff=None*, *grompp_check=False*)

gropath = path to gromacs binaries grosuff = suffix of gromacs binaries, usually '' or '_d'

intermol.gromacs.gromacs_driver.**read_file**(*top_in*, *gro_in*, *gropath*)

intermol.gromacs.gromacs_driver.**write_file**(*system*, *top_out*, *gro_out*)

intermol.gromacs.gromacs_parser module

class intermol.gromacs.gromacs_parser.**GromacsParser**(*top_file*, *gro_file*, *system=None*, *include_dir=None*, *defines=None*)

Bases: object

A class containing methods required to read in a Gromacs(4.5.4) Topology File

class TopMoleculeType

Bases: object

Inner class to store information about a molecule type.

GromacsParser.**canonical_angle**(*params*, *angle*, *direction='into'*)

Parameters

- **params** –
- **angle** –
- **direction** –

Returns:

GromacsParser.**canonical_bond**(*params*, *bond*, *direction='into'*)

Parameters

- **params** –
- **bond** –
- **direction** –

Returns:

GromacsParser.**canonical_dihedral**(*params*, *dihedral*, *direction='into'*)

We can fit everything into two types of dihedrals - dihedral_trig, and improper harmonic. Dihedral trig is of the form

fc0 + sum_i=1^6 fci (cos(nx-phi)

Proper dihedrals can be stored easily in this form, since they have only 1 n. Improper dihedrals can as well (flag as improper). RB can be stored as well, assuming phi = 0 or 180. Fourier can also be stored. A full dihedral trig can be decomposed into multiple proper dihedrals.

Will need to handle multiple dihedrals little differently in that we will need to add multiple 9 dihedrals together into a single dihedral_trig, as long as they have the same phi angle (seems to be always the case).

Parameters

- **params** –
- **dihedral** –
- **direction** –

Returns:

GromacsParser.**choose_parameter_kwds_from_forces**(*entries*, *n_atoms*, *force_type*, *gromacs_force*)

Extract a force's parameters into a keyword dictionary.

Parameters

- **entries** (*str*) – The *split()* line being parsed.
- **n_atoms** (*int*) – The number of atoms in the force.
- **force_type** – The type of the force.
- **gromacs_force** – The

Returns

kwds – The force's parameters, e.g.

{'length': Quantity(value=0.13, unit=nanometers), 'k': ...

}

Return type dict

GromacsParser.**create_angle**(*angle*)

GromacsParser.**create_atom**(*temp_atom*)

GromacsParser.**create_bond**(*bond*)

GromacsParser.**create_dihedral**(*dihedral*)

Create a dihedral object based on a [dihedrals] entry.

GromacsParser.**create_exclusion**(*exclusion*)

GromacsParser.**create_kwds_from_entries**(*entries*, *force_class*, *offset=0*)

GromacsParser.**create_molecule**(*top_moltype*, *mol_name*)

GromacsParser.**create_moleculetype**(*top_moltype*, *mol_name*, *mol_count*)

GromacsParser.**create_pair**(*pair*)

Create a pair force object based on a [pairs] entry

GromacsParser.**create_settle**(*settle*)

GromacsParser.**directive_before_moleculetype**()

GromacsParser.**find_dihedraltype**(*bondingtypes*, *improper*)

Determine the type of dihedral interaction between four atoms.

GromacsParser.**find_forcetype**(*bondingtypes*, *types_of_kind*)

GromacsParser.**get_parameter_kwds_from_force**(*force*)

GromacsParser.**get_parameter_list_from_force**(*force*)

GromacsParser.**gromacs_angle_types = {'1': <class 'intermol.forces.harmonic_angle_type.HarmonicAngleType'**

GromacsParser.**gromacs_angles = {'1': <class 'intermol.forces.harmonic_angle_type.HarmonicAngle'>, '3': <class**

GromacsParser.**gromacs_bond_types = {'1': <class 'intermol.forces.harmonic_bond_type.HarmonicBondType'>,**

GromacsParser.**gromacs_bonds = {'1': <class 'intermol.forces.harmonic_bond_type.HarmonicBond'>, '3': <class '**

GromacsParser.**gromacs_combination_rules = {'1': 'Multiply-C6C12', '3': 'Multiply-Sigeps', '2': 'Lorentz-Ber**

GromacsParser.**gromacs_dihedral_types = {'Trig': <class 'intermol.forces.trig_dihedral_type.TrigDihedralType**

GromacsParser.**gromacs_dihedrals = {'Trig': <class 'intermol.forces.trig_dihedral_type.TrigDihedral'>, '1': <cla**

GromacsParser.**gromacs_pair_types = {'1A': <class 'intermol.forces.lj_c_pair_type.LjCPairType'>, '1C': <class**

GromacsParser.**gromacs_pairs = {'1A': <class 'intermol.forces.lj_c_pair_type.LjCPair'>, '1C': <class 'intermol.fo**

GromacsParser.**invalid_line**(*line*)

GromacsParser.**lookup_atom_atomtype**(*index*, *state=0*)

GromacsParser.**lookup_atom_bondingtype**(*index*)

GromacsParser.**lookup_gromacs_angles = {<class 'intermol.forces.cross_bond_bond_angle_type.CrossBondBon**

GromacsParser.**lookup_gromacs_bonds = {<class 'intermol.forces.fene_bond_type.FeneBond'>: '7', <class 'inter**

GromacsParser.**lookup_gromacs_combination_rules = {'Multiply-Sigeps': '3', 'Lorentz-Berthelot': '2', 'Mul**

GromacsParser.**lookup_gromacs_dihedrals = {<class 'intermol.forces.fourier_dihedral_type.FourierDihedral'>**

GromacsParser.**lookup_gromacs_pairs = {<class 'intermol.forces.ljq_sigeps_pair_type.LjqSigepsPair'>: '2B', <c**

GromacsParser.**paramlist = {'QuarticBreakableBond': ['k', 'B1', 'B2', 'Rc', 'U0'], 'fene_expandable_bond': ['k', '**

GromacsParser.**process_angle**(*line*)
Process a line in the [angles] category.

GromacsParser.**process_angletype**(*line*)
Process a line in the [angletypes] category.

GromacsParser.**process_atom**(*line*)
Process a line in the [atoms] category.

GromacsParser.**process_atomtype**(*line*)
Process a line in the [atomtypes] category.

GromacsParser.**process_bond**(*line*)
Process a line in the [bonds] category.

GromacsParser.**process_bondtype**(*line*)
Process a line in the [bondtypes] category.

GromacsParser.**process_cmap**(*line*)
Process a line in the [cmaps] category.

GromacsParser.**process_cmaptype**(*line*)
Process a line in the [cmaptypes] category.

GromacsParser.**process_defaults**(*line*)
Process the [defaults] line.

GromacsParser.**process_dihedral**(*line*)
Process a line in the [dihedrals] category.

GromacsParser.**process_dihedraltype**(*line*)
Process a line in the [dihedraltypes] category.

GromacsParser.**process_exclusion**(*line*)
Process a line in the [exclusions] category.

GromacsParser.**process_file**(*top_file*)

GromacsParser.**process_forcetype**(*bondingtypes*, *forcename*, *line*, *n_atoms*, *gromacs_force_types*, *canonical_force*)

GromacsParser.**process_implicittype**(*line*)
Process a line in the [implicit_genborn_params] category.

GromacsParser.**process_line**(*top_file*, *line*)
Process one line from a file.

GromacsParser.**process_molecule**(*line*)
Process a line in the [molecules] category.

GromacsParser.**process_moleculetype**(*line*)
Process a line in the [moleculetypes] category.

GromacsParser.**process_nonbond_params**(*line*)
Process a line in the [nonbond_param] category.

GromacsParser.**process_pair**(*line*)
Process a line in the [pairs] category.

GromacsParser.**process_pairtype**(*line*)
Process a line in the [pairtypes] category.

GromacsParser.**process_settle**(*line*)
Process a line in the [settles] category.

GromacsParser.**read**()

Returns system

GromacsParser.**too_few_fields**(*line*)

static GromacsParser.**type_parameters_are_unique**(*a*, *b*)
Check if two force types are unique.

Currently only tests TrigDihedralType and ImproperHarmonicDihedralType because these are the only two forcetypes that we currently allow to to have multiple values for the same set of 4 atom bondingtypes.

GromacsParser.**unitvars = {'QuarticBreakableBond': [Unit({BaseUnit(base_dim=BaseDimension("length"), name=**

GromacsParser.**write**()
Write this topology in GROMACS file format.

Parameters **filename** – the name of the file to write out to

GromacsParser.**write_angles**(*top*)

GromacsParser.**write_atoms**(*top*)

GromacsParser.**write_atomtypes**(*top*)

GromacsParser.**write_bonds**(*top*)

GromacsParser.**write_defaults**(*top*)

GromacsParser.**write_dihedrals**(*top*)

GromacsParser.**write_exclusions**(*top*)

GromacsParser.**write_molecules**(*top*)

GromacsParser.**write_moleculetypes**(*top*)

GromacsParser.**write_nonbonded_types**(*top*)

GromacsParser.**write_pairs**(*top*)

GromacsParser.**write_settles**(*top*)

GromacsParser.**write_system**(*top*)

intermol.gromacs.gromacs_parser.**default_gromacs_include_dir**()
Find the location where gromacs #include files are referenced from, by searching for (1) gromacs environment variables, (2) just using the default gromacs install location, /usr/local/gromacs/share/gromacs/top.

intermol.gromacs.gromacs_parser.**load_gromacs**(*top_file*, *gro_file*, *include_dir=None*, *defines=None*)
Load a set of GROMACS input files into a *System*.

Parameters

- **top_file** –
- **gro_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

intermol.gromacs.gromacs_parser.**write_gromacs**(*top_file*, *gro_file*, *system*)
Load a set of GROMACS input files into a *System*.

Parameters

- **top_file** –
- **gro_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

Module contents

intermol.lammps package

Submodules

intermol.lammps.lammps_driver module

intermol.lammps.lammps_driver.**lammps_energies**(*input_file*, *lmppath='lmp_openmpi'*)
Evaluate energies of LAMMPS files

Parameters

- **= path to input file (expects data file in same folder)** (*input_file*) –
- **= path to LAMMPS binaries** (*lmppath*) –

intermol.lammps.lammps_driver.**read_file**(*in_file*)

intermol.lammps.lammps_driver.**write_file**(*in_file*, *system*, *unit_set='real'*)

intermol.lammps.lammps_parser module

class intermol.lammps.lammps_parser.**LammpsParser**(*in_file*, *system=None*, *unit_set='real'*)
Bases: object

A class containing methods to read and write LAMMPS files.

SCALE_FROM = 0.5

SCALE_INTO = 2.0

canonical_angle(*kwds*, *angle*, *direction*)
Convert from the canonical form of this interaction.

canonical_bond(*kwds*, *bond*, *direction='into'*)
Convert to/from the canonical form of this interaction.

canonical_dihedral(*params*, *dihedral*, *direction='into'*)
Convert from the canonical form of this interaction.

create_kwds_from_entries(*entries*, *force_class*, *offset=0*)

get_force_atoms(*force*, *forceclass*)
Return the atoms involved in a force.

get_force_bondingtypes(*force*, *forceclass*)
Return the atoms involved in a force.

get_parameter_kwds_from_force(*force*)

get_parameter_list_from_force(*force*)

lammps_angle_types = {'charmm': <class 'intermol.forces.urey_bradley_angle_type.UreyBradleyAngleType'>, 'cosi

lammps_angles = {'charmm': <class 'intermol.forces.urey_bradley_angle_type.UreyBradleyAngle'>, 'cosine': <class 'i

lammps_bond_types = {'nonlinear': <class 'intermol.forces.nonlinear_bond_type.NonlinearBondType'>, 'quartic': <c

lammps_bonds = {'nonlinear': <class 'intermol.forces.nonlinear_bond_type.NonlinearBond'>, 'quartic': <class 'interm

lammps_dihedral_types = {'charmm': <class 'intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedr

lammps_dihedrals = {'charmm': <class 'intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedral'>, 'r

lammps_improper_types = {'cvff': <class 'intermol.forces.trig_dihedral_type.TrigDihedralType'>, 'harmonic': <clas

lammps_impropers = {'cvff': <class 'intermol.forces.trig_dihedral_type.TrigDihedral'>, 'harmonic': <class 'intermol.

lookup_lammps_angles = {<class 'intermol.forces.cosine_angle_type.CosineAngle'>: 'cosine', <class 'intermol.forces.

lookup_lammps_bonds = {<class 'intermol.forces.harmonic_bond_type.HarmonicBond'>: 'harmonic', <class 'interm

lookup_lammps_dihedrals = {<class 'intermol.forces.fourier_dihedral_type.FourierDihedral'>: 'opls', <class 'inter

lookup_lammps_impropers = {<class 'intermol.forces.improper_harmonic_dihedral_type.ImproperHarmonicDihedr

parse_angle_coeffs(*data_lines*)

parse_angle_style(*line*)

parse_angles(*data_lines*)

parse_atom_style(*line*)

Note: Assuming 'full' as default for everything else.

---

parse_atoms(*data_lines*)
Read atoms from data file.

parse_bond_coeffs(*data_lines*)

parse_bond_style(*line*)

parse_bonded_style(*line*)

parse_bonds(*data_lines*)

parse_boundary(*line*)

parse_box(*line*, *dim*)
Read box information from data file.

Parameters

- **line** (*str*) – Current line in input file.
- **dim** (*int*) – Dimension specified in line.

parse_dihedral_coeffs(*data_lines*)

parse_dihedral_style(*line*)

parse_dihedrals(*data_lines*)

parse_dimension(*line*)

parse_force(*data_lines*, *force_classes*, *forceSet*, *n=0*)
Read bonds, angles, dihedrals, impropers from data file.

parse_force_coeffs(*data_lines*, *force_name*, *force_classes*, *force_style*, *lammps_forces*, *canonical_force*)
Read force coefficients from data file.

parse_improper_coeffs(*data_lines*)

parse_improper_style(*line*)

parse_impropers(*data_lines*)

parse_kspace_style(*line*)

Note: Currently ignored.

---

parse_masses(*data_lines*)
Read masses from data file.

parse_pair_coeffs(*data_lines*)
Read pair coefficients from data file.

parse_pair_modify(*line*)

parse_pair_style(*line*)

parse_read_data(*line*)

parse_special_bonds(*line*)

parse_units(*line*)

parse_velocities(*data_lines*)

read()
Reads a LAMMPS input file and a data file specified within.

Parameters **input_file** (*str*) – Name of LAMMPS input file to read in.

read_data(*data_file*)
Reads a LAMMPS data file.

Parameters **data_file** (*str*) – name of LAMMPS data file to read in.

read_input()
Reads a LAMMPS input file.

Parameters **input_file** (*str*) – Name of LAMMPS input file to read in.

set_units(*unit_set*)
Set what unit set to use.

write(*unit_set='real'*)
Writes a LAMMPS data and corresponding input file.

Parameters

- **data_file** (*str*) – Name of LAMMPS data file to write to.
- **unit_set** (*str*) – LAMMPS unit set for output file.

write_angles(*mol_type*, *offset*)

write_bonds(*mol_type*, *offset*)

write_dihedrals(*mol_type*, *offset*)
Separate dihedrals from impropers.

write_forces(*forces*, *offset*, *force_name*, *lookup_lammps_force*, *lammps_force_types*, *canonical_force*)
The general force writing function.

Currently supports bonds, angles, dihedrals, impropers.

write_impropers(*mol_type*, *offset*)
Separate dihedrals from impropers.

write_virtuals(*mol_type*, *offset*)

intermol.lammps.lammps_parser.**load_lammps**(*in_file*)
Load a LAMMPS input file into a *System*.

Parameters

- **in_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

intermol.lammps.lammps_parser.**write_lammps**(*in_file*, *system*, *unit_set='real'*)
Load a LAMMPS input file into a *System*.

Parameters

- **in_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

Module contents

3.1.2 Submodules

intermol.atom module

class intermol.atom.**Atom**(*index*, *name=None*, *residue_index=-1*, *residue_name=None*)
Bases: object

atomic_number

atomtype

bondingtype

cgnr

charge

epsilon

force
Return the force on the atom

index

mass

name

position
Return the cartesian coordinates of the atom

ptype

residue_index

residue_name

sigma

velocity
Return the velocity of the atom

intermol.convert module

intermol.convert.**find_match**(*key*, *dictionary*, *unit*)
Helper function for *summarize_energy_results*.

intermol.convert.**get_diff**(*e_in*, *e_out*)
Returns difference in potential energy.

Parameters

- **- dictionary of energy groups from input file** (*e_in*) –
- **- dictionary of energy groups from output file** (*e_out*) –

Returns potential energy difference in units of the input

intermol.convert.**main**(*args=None*)

intermol.convert.**parse_args**(*args*)

intermol.convert.**summarize_energy_results**(*energy_input*, *energy_outputs*, *input_type*, *output_types*)

Creates a table comparing input and output energy groups.

Args: energy_input (dict): energy groups from input file energy_output(list): containing dictionary of energy groups or -1 for

each output file

input_type (str): input engine output_types (list): containing output formats

Returns: out (list of strings): which forms a summary table using "

".join(out)

intermol.decorators module

exception intermol.decorators.**UnitsException**(*value*)

Bases: exceptions.Exception

Exception denoting that an argument has the incorrect units.

exception intermol.decorators.**ValueException**(*value*)

Bases: exceptions.Exception

Exception denoting that an argument has the incorrect value.

intermol.decorators.**accepts**(**types*)

Decorator for class methods that should accept only specified types.

EXAMPLE

@accepts(float, int) def function(a, b):

return b*a

intermol.decorators.**accepts_compatible_units**(**units*, ***unitdict*)

Decorator for class methods that should accept only arguments compatible with specified units.

Each argument of the function will be matched with an argument of @acceptunits. Those arguments of the function that correspond @acceptunits which are not None will be checked to ensure they are compatible with the specified units.

EXAMPLE

@acceptsunits(units.meter, None, units.kilocalories_per_mole) def function(a, b, c): pass function(1.0 * units.angstrom, 3, 1.0 * units.kilojoules_per_mole)

intermol.decorators.**returns**(*rtype*)

Decorator for functions that should only return specific types. EXAMPLE

@returns(int) def function(): return 7

intermol.molecule module

class intermol.molecule.**Molecule**(*name=None*)

Bases: object

An abstract molecule object.

add_atom(*atom*)

Add an atom

Parameters **atom** (*Atom*) – the atom to add into the molecule

atoms
Return an orderedset of atoms.

intermol.moleculetype module

class intermol.moleculetype.**MoleculeType**(*name=None*)
Bases: object

An abstract container for molecules of one type.

add_molecule(*molecule*)
Add a molecule into the moleculetype container

Parameters **molecule** (*Molecule*) – the molecule to append

intermol.orderedset module

class intermol.orderedset.**OrderedSet**(*iterable=()*)
Bases: _abcoll.Set

add(*key*)

difference_update(**args*, ***kwargs*)

discard(*key*)

intersection_update(**args*, ***kwargs*)

intermol.system module

class intermol.system.**System**(*name=None*)
Bases: object

add_atomtype(*atomtype*)

add_molecule(*molecule*)
Append a molecule into the System. :param molecule: The molecule object to be appended

add_molecule_type(*molecule_type*)
Append a molecule_type into the System. :param molecule_type: The MoleculeType object to be appended

atoms

atomtypes

box_vector
Return the box vector.

molecule_types

n_atoms

nonbonded_types

3.1.3 Module contents

3.2 intermol.gromacs package

3.2.1 Submodules

intermol.gromacs.grofile_parser module

class intermol.gromacs.grofile_parser.**GromacsGroParser**(*gro_file*)
Bases: object

GromacsGroParser reads and writes Gromacs .gro files

A .gro file also contains some topological information, such as elements and residue names, but not enough to construct a full Topology object. This information is recorded and stored in the object's public fields.

read()

write(*system*)
Write the system out in a Gromacs 4.6 format

Parameters **filename** (*str*) – the file to write out to

intermol.gromacs.gromacs_driver module

intermol.gromacs.gromacs_driver.**gromacs_energies**(*top=None*, *gro=None*, *mdp=None*, *gropath=None*, *grosuff=None*, *grompp_check=False*)
gropath = path to gromacs binaries grosuff = suffix of gromacs binaries, usually '' or '_d'

intermol.gromacs.gromacs_driver.**read_file**(*top_in*, *gro_in*, *gropath*)

intermol.gromacs.gromacs_driver.**write_file**(*system*, *top_out*, *gro_out*)

intermol.gromacs.gromacs_parser module

class intermol.gromacs.gromacs_parser.**GromacsParser**(*top_file*, *gro_file*, *system=None*, *include_dir=None*, *defines=None*)
Bases: object

A class containing methods required to read in a Gromacs(4.5.4) Topology File

class TopMoleculeType
Bases: object

Inner class to store information about a molecule type.

GromacsParser.**canonical_angle**(*params*, *angle*, *direction='into'*)

Parameters

- **params** –
- **angle** –
- **direction** –

Returns:

GromacsParser.**canonical_bond**(*params*, *bond*, *direction='into'*)

Parameters

- **params** –
- **bond** –
- **direction** –

Returns:

GromacsParser.**canonical_dihedral**(*params*, *dihedral*, *direction='into'*)

We can fit everything into two types of dihedrals - dihedral_trig, and improper harmonic. Dihedral trig is of the form

fc0 + sum_i=1^6 fci (cos(nx-phi)

Proper dihedrals can be stored easily in this form, since they have only 1 n. Improper dihedrals can as well (flag as improper). RB can be stored as well, assuming phi = 0 or 180. Fourier can also be stored. A full dihedral trig can be decomposed into multiple proper dihedrals.

Will need to handle multiple dihedrals little differently in that we will need to add multiple 9 dihedrals together into a single dihedral_trig, as long as they have the same phi angle (seems to be always the case).

Parameters

- **params** –
- **dihedral** –
- **direction** –

Returns:

GromacsParser.**choose_parameter_kwds_from_forces**(*entries*, *n_atoms*, *force_type*, *gromacs_force*)

Extract a force's parameters into a keyword dictionary.

Parameters

- **entries** (*str*) – The *split()* line being parsed.
- **n_atoms** (*int*) – The number of atoms in the force.
- **force_type** – The type of the force.
- **gromacs_force** – The

Returns

kwds – The force's parameters, e.g.

{'length': Quantity(value=0.13, unit=nanometers), 'k': ...

}

Return type dict

GromacsParser.**create_angle**(*angle*)

GromacsParser.**create_atom**(*temp_atom*)

GromacsParser.**create_bond**(*bond*)

GromacsParser.**create_dihedral**(*dihedral*)

Create a dihedral object based on a [dihedrals] entry.

GromacsParser.**create_exclusion**(*exclusion*)

GromacsParser.**create_kwds_from_entries**(*entries*, *force_class*, *offset=0*)

GromacsParser.**create_molecule**(*top_moltype*, *mol_name*)

GromacsParser.**create_moleculetype**(*top_moltype*, *mol_name*, *mol_count*)

GromacsParser.**create_pair**(*pair*)
Create a pair force object based on a [pairs] entry

GromacsParser.**create_settle**(*settle*)

GromacsParser.**directive_before_moleculetype**()

GromacsParser.**find_dihedraltype**(*bondingtypes*, *improper*)
Determine the type of dihedral interaction between four atoms.

GromacsParser.**find_forcetype**(*bondingtypes*, *types_of_kind*)

GromacsParser.**get_parameter_kwds_from_force**(*force*)

GromacsParser.**get_parameter_list_from_force**(*force*)

GromacsParser.**gromacs_angle_types** = {'1': <class 'intermol.forces.harmonic_angle_type.HarmonicAngleType'>

GromacsParser.**gromacs_angles** = {'1': <class 'intermol.forces.harmonic_angle_type.HarmonicAngle'>, '3': <class

GromacsParser.**gromacs_bond_types** = {'1': <class 'intermol.forces.harmonic_bond_type.HarmonicBondType'>,

GromacsParser.**gromacs_bonds** = {'1': <class 'intermol.forces.harmonic_bond_type.HarmonicBond'>, '3': <class '

GromacsParser.**gromacs_combination_rules** = {'1': 'Multiply-C6C12', '3': 'Multiply-Sigeps', '2': 'Lorentz-Ber

GromacsParser.**gromacs_dihedral_types** = {'Trig': <class 'intermol.forces.trig_dihedral_type.TrigDihedralType

GromacsParser.**gromacs_dihedrals** = {'Trig': <class 'intermol.forces.trig_dihedral_type.TrigDihedral'>, '1': <cla

GromacsParser.**gromacs_pair_types** = {'1A': <class 'intermol.forces.lj_c_pair_type.LjCPairType'>, '1C': <class

GromacsParser.**gromacs_pairs** = {'1A': <class 'intermol.forces.lj_c_pair_type.LjCPair'>, '1C': <class 'intermol.fo

GromacsParser.**invalid_line**(*line*)

GromacsParser.**lookup_atom_atomtype**(*index*, *state=0*)

GromacsParser.**lookup_atom_bondingtype**(*index*)

GromacsParser.**lookup_gromacs_angles** = {<class 'intermol.forces.cross_bond_bond_angle_type.CrossBondBond

GromacsParser.**lookup_gromacs_bonds** = {<class 'intermol.forces.fene_bond_type.FeneBond'>: '7', <class 'interm

GromacsParser.**lookup_gromacs_combination_rules** = {'Multiply-Sigeps': '3', 'Lorentz-Berthelot': '2', 'Mult

GromacsParser.**lookup_gromacs_dihedrals** = {<class 'intermol.forces.fourier_dihedral_type.FourierDihedral'>

GromacsParser.**lookup_gromacs_pairs** = {<class 'intermol.forces.ljq_sigeps_pair_type.LjqSigepsPair'>: '2B', <c

GromacsParser.**paramlist** = {'QuarticBreakableBond': ['k', 'B1', 'B2', 'Rc', 'U0'], 'fene_expandable_bond': ['k', '

GromacsParser.**process_angle**(*line*)
Process a line in the [angles] category.

GromacsParser.**process_angletype**(*line*)
Process a line in the [angletypes] category.

GromacsParser.**process_atom**(*line*)
Process a line in the [atoms] category.

GromacsParser.**process_atomtype**(*line*)
Process a line in the [atomtypes] category.

GromacsParser.**process_bond**(*line*)
Process a line in the [bonds] category.

GromacsParser.**process_bondtype**(*line*)
Process a line in the [bondtypes] category.

GromacsParser.**process_cmap**(*line*)
Process a line in the [cmaps] category.

GromacsParser.**process_cmaptype**(*line*)
Process a line in the [cmaptypes] category.

GromacsParser.**process_defaults**(*line*)
Process the [defaults] line.

GromacsParser.**process_dihedral**(*line*)
Process a line in the [dihedrals] category.

GromacsParser.**process_dihedraltype**(*line*)
Process a line in the [dihedraltypes] category.

GromacsParser.**process_exclusion**(*line*)
Process a line in the [exclusions] category.

GromacsParser.**process_file**(*top_file*)

GromacsParser.**process_forcetype**(*bondingtypes*, *forcename*, *line*, *n_atoms*, *gromacs_force_types*, *canonical_force*)

GromacsParser.**process_implicittype**(*line*)
Process a line in the [implicit_genborn_params] category.

GromacsParser.**process_line**(*top_file*, *line*)
Process one line from a file.

GromacsParser.**process_molecule**(*line*)
Process a line in the [molecules] category.

GromacsParser.**process_moleculetype**(*line*)
Process a line in the [moleculetypes] category.

GromacsParser.**process_nonbond_params**(*line*)
Process a line in the [nonbond_param] category.

GromacsParser.**process_pair**(*line*)
Process a line in the [pairs] category.

GromacsParser.**process_pairtype**(*line*)
Process a line in the [pairtypes] category.

GromacsParser.**process_settle**(*line*)
Process a line in the [settles] category.

GromacsParser.**read**()

Returns system

GromacsParser.**too_few_fields**(*line*)

static GromacsParser.**type_parameters_are_unique**(*a*, *b*)
Check if two force types are unique.

Currently only tests TrigDihedralType and ImproperHarmonicDihedralType because these are the only two forcetypes that we currently allow to to have multiple values for the same set of 4 atom bondingtypes.

GromacsParser.**unitvars = {'QuarticBreakableBond': [Unit({BaseUnit(base_dim=BaseDimension("length"), name=**

GromacsParser.**write**()
Write this topology in GROMACS file format.

Parameters **filename** – the name of the file to write out to

GromacsParser.**write_angles**(*top*)

GromacsParser.**write_atoms**(*top*)

GromacsParser.**write_atomtypes**(*top*)

GromacsParser.**write_bonds**(*top*)

GromacsParser.**write_defaults**(*top*)

GromacsParser.**write_dihedrals**(*top*)

GromacsParser.**write_exclusions**(*top*)

GromacsParser.**write_molecules**(*top*)

GromacsParser.**write_moleculetypes**(*top*)

GromacsParser.**write_nonbonded_types**(*top*)

GromacsParser.**write_pairs**(*top*)

GromacsParser.**write_settles**(*top*)

GromacsParser.**write_system**(*top*)

intermol.gromacs.gromacs_parser.**default_gromacs_include_dir**()
Find the location where gromacs #include files are referenced from, by searching for (1) gromacs environment variables, (2) just using the default gromacs install location, /usr/local/gromacs/share/gromacs/top.

intermol.gromacs.gromacs_parser.**load_gromacs**(*top_file*, *gro_file*, *include_dir=None*, *defines=None*)
Load a set of GROMACS input files into a *System*.

Parameters

- **top_file** –
- **gro_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

intermol.gromacs.gromacs_parser.**write_gromacs**(*top_file*, *gro_file*, *system*)
Load a set of GROMACS input files into a *System*.

Parameters

- **top_file** –
- **gro_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

3.2.2 Module contents

3.3 intermol.lammps package

3.3.1 Submodules

intermol.lammps.lammps_driver module

intermol.lammps.lammps_driver.**lammps_energies**(*input_file*, *lmppath='lmp_openmpi'*)
Evaluate energies of LAMMPS files

Parameters

- **= path to input file (expects data file in same folder)** (*input_file*) –
- **= path to LAMMPS binaries** (*lmppath*) –

intermol.lammps.lammps_driver.**read_file**(*in_file*)

intermol.lammps.lammps_driver.**write_file**(*in_file*, *system*, *unit_set='real'*)

intermol.lammps.lammps_parser module

class intermol.lammps.lammps_parser.**LammpsParser**(*in_file*, *system=None*, *unit_set='real'*)
Bases: object

A class containing methods to read and write LAMMPS files.

SCALE_FROM = 0.5

SCALE_INTO = 2.0

canonical_angle(*kwds*, *angle*, *direction*)
Convert from the canonical form of this interaction.

canonical_bond(*kwds*, *bond*, *direction='into'*)
Convert to/from the canonical form of this interaction.

canonical_dihedral(*params*, *dihedral*, *direction='into'*)
Convert from the canonical form of this interaction.

create_kwds_from_entries(*entries*, *force_class*, *offset=0*)

get_force_atoms(*force*, *forceclass*)
Return the atoms involved in a force.

get_force_bondingtypes(*force*, *forceclass*)
Return the atoms involved in a force.

get_parameter_kwds_from_force(*force*)

get_parameter_list_from_force(*force*)

lammps_angle_types = {'charmm': <class 'intermol.forces.urey_bradley_angle_type.UreyBradleyAngleType'>, 'cosi

lammps_angles = {'charmm': <class 'intermol.forces.urey_bradley_angle_type.UreyBradleyAngle'>, 'cosine': <class 'i

lammps_bond_types = {'nonlinear': <class 'intermol.forces.nonlinear_bond_type.NonlinearBondType'>, 'quartic': <c

lammps_bonds = {'nonlinear': <class 'intermol.forces.nonlinear_bond_type.NonlinearBond'>, 'quartic': <class 'interm

lammps_dihedral_types = {'charmm': <class 'intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedr

lammps_dihedrals = {'charmm': <class 'intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedral'>, 'r

lammps_improper_types = {'cvff': <class 'intermol.forces.trig_dihedral_type.TrigDihedralType'>, 'harmonic': <clas

lammps_impropers = {'cvff': <class 'intermol.forces.trig_dihedral_type.TrigDihedral'>, 'harmonic': <class 'intermol.

lookup_lammps_angles = {<class 'intermol.forces.cosine_angle_type.CosineAngle'>: 'cosine', <class 'intermol.forces.

lookup_lammps_bonds = {<class 'intermol.forces.harmonic_bond_type.HarmonicBond'>: 'harmonic', <class 'interm

lookup_lammps_dihedrals = {<class 'intermol.forces.fourier_dihedral_type.FourierDihedral'>: 'opls', <class 'inter

lookup_lammps_impropers = {<class 'intermol.forces.improper_harmonic_dihedral_type.ImproperHarmonicDihed

parse_angle_coeffs(*data_lines*)

parse_angle_style(*line*)

parse_angles(*data_lines*)

parse_atom_style(*line*)

> **Note:** Assuming 'full' as default for everything else.

parse_atoms(*data_lines*)
Read atoms from data file.

parse_bond_coeffs(*data_lines*)

parse_bond_style(*line*)

parse_bonded_style(*line*)

parse_bonds(*data_lines*)

parse_boundary(*line*)

parse_box(*line*, *dim*)
Read box information from data file.

Parameters

- **line** (*str*) – Current line in input file.
- **dim** (*int*) – Dimension specified in line.

parse_dihedral_coeffs(*data_lines*)

parse_dihedral_style(*line*)

parse_dihedrals(*data_lines*)

parse_dimension(*line*)

parse_force(*data_lines*, *force_classes*, *forceSet*, *n=0*)
Read bonds, angles, dihedrals, impropers from data file.

parse_force_coeffs(*data_lines*, *force_name*, *force_classes*, *force_style*, *lammps_forces*, *canonical_force*)
Read force coefficients from data file.

parse_improper_coeffs(*data_lines*)

parse_improper_style(*line*)

parse_impropers(*data_lines*)

parse_kspace_style(*line*)

> **Note:** Currently ignored.

parse_masses(*data_lines*)
Read masses from data file.

parse_pair_coeffs(*data_lines*)
Read pair coefficients from data file.

parse_pair_modify(*line*)

parse_pair_style(*line*)

parse_read_data(*line*)

parse_special_bonds(*line*)

parse_units(*line*)

parse_velocities(*data_lines*)

read()
Reads a LAMMPS input file and a data file specified within.

Parameters **input_file** (*str*) – Name of LAMMPS input file to read in.

read_data(*data_file*)
Reads a LAMMPS data file.

Parameters **data_file** (*str*) – name of LAMMPS data file to read in.

read_input()
Reads a LAMMPS input file.

Parameters **input_file** (*str*) – Name of LAMMPS input file to read in.

set_units(*unit_set*)
Set what unit set to use.

write(*unit_set='real'*)
Writes a LAMMPS data and corresponding input file.

Parameters

- **data_file** (*str*) – Name of LAMMPS data file to write to.
- **unit_set** (*str*) – LAMMPS unit set for output file.

write_angles(*mol_type*, *offset*)

write_bonds(*mol_type*, *offset*)

write_dihedrals(*mol_type*, *offset*)
Separate dihedrals from impropers.

write_forces(*forces*, *offset*, *force_name*, *lookup_lammps_force*, *lammps_force_types*, *canonical_force*)
The general force writing function.

Currently supports bonds, angles, dihedrals, impropers.

write_impropers(*mol_type*, *offset*)
Separate dihedrals from impropers.

write_virtuals(*mol_type*, *offset*)

intermol.lammps.lammps_parser.**load_lammps**(*in_file*)
Load a LAMMPS input file into a *System*.

Parameters

- **in_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

intermol.lammps.lammps_parser.**write_lammps**(*in_file*, *system*, *unit_set='real'*)
Load a LAMMPS input file into a *System*.

Parameters

- **in_file** –
- **include_dir** –
- **defines** –

Returns

Return type system

3.3.2 Module contents

3.4 intermol.forces package

3.4.1 Submodules

intermol.forces.abstract_2_virtual_type module

class intermol.forces.abstract_2_virtual_type.**Abstract2VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*)
Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

placeholder

intermol.forces.abstract_3_virtual_type module

class intermol.forces.abstract_3_virtual_type.**Abstract3VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

placeholder

intermol.forces.abstract_4_virtual_type module

class intermol.forces.abstract_4_virtual_type.**Abstract4VirtualType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*, *bondingtype5*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

intermol.forces.abstract_angle_type module

class intermol.forces.abstract_angle_type.**AbstractAngleType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *c=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

c

intermol.forces.abstract_atom_type module

class intermol.forces.abstract_atom_type.**AbstractAtomType**(*atomtype*, *bondtype=None*, *atomic_number=None*, *mass=None*, *charge=None*, *ptype=None*)

Bases: intermol.forces.abstract_type.AbstractType

atomic_number

atomtype

bondtype

charge

mass

ptype

intermol.forces.abstract_bond_type module

class intermol.forces.abstract_bond_type.**AbstractBondType**(*bondingtype1*, *bondingtype2*, *order=1*, *c=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

c

order

intermol.forces.abstract_dihedral_type module

class intermol.forces.abstract_dihedral_type.**AbstractDihedralType**(*bondingtype1*, *bondingtype2*, *bondingtype3*, *bondingtype4*, *improper=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

bondingtype3

bondingtype4

improper

intermol.forces.abstract_nonbonded_type module

class intermol.forces.abstract_nonbonded_type.**AbstractNonbondedType**(*atom1*, *atom2*, *type*)

Bases: intermol.forces.abstract_type.AbstractType

atom1

atom2

type

intermol.forces.abstract_pair_type module

class intermol.forces.abstract_pair_type.**AbstractPairType**(*bondingtype1*, *bondingtype2*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.abstract_type.AbstractType

bondingtype1

bondingtype2

long

scaleLJ

scaleQQ

intermol.forces.abstract_type module

class intermol.forces.abstract_type.**AbstractType**

Bases: object

__repr__()

Print the object and all of its non-magic attributes.

intermol.forces.atom_c_type module

class intermol.forces.atom_c_type.**AtomCType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_atom_type.AbstractAtomType

intermol.forces.atom_sigeps_type module

class intermol.forces.atom_sigeps_type.**AtomSigepsType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_atom_type.AbstractAtomType

intermol.forces.buckingham_nonbonded_type module

class intermol.forces.buckingham_nonbonded_type.**BuckinghamNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *a=Quantity(value=0.0, unit=kilojoule/mole)*, *b=Quantity(value=0.0, unit=/nanometer)*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *type=False*)

Bases: intermol.forces.buckingham_nonbonded_type.BuckinghamNonbondedType

stub documentation

class intermol.forces.buckingham_nonbonded_type.**BuckinghamNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

C6

a

b

type

intermol.forces.connection_bond_type module

class intermol.forces.connection_bond_type.**ConnectionBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *bondingorder=1*, *c=False*)

Bases: intermol.forces.connection_bond_type.ConnectionBondType

stub documentation

class intermol.forces.connection_bond_type.**ConnectionBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

order

intermol.forces.convert_dihedrals module

intermol.forces.convert_dihedrals.**convert_dihedral_from_OPLS_to_RB**(*f*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_RB_to_OPLS**(*c*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_RB_to_trig**(*c*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_fourier_to_trig**(*f*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_proper_to_trig**(*p*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_RB**(*fcs*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_fourier**(*fcs*)

intermol.forces.convert_dihedrals.**convert_dihedral_from_trig_to_proper**(*fcs*, *convention='0'*)

intermol.forces.convert_dihedrals.**convert_nothing**(*x*)
useful utility for not converting anything

intermol.forces.cosine_angle_type module

class intermol.forces.cosine_angle_type.**CosineAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *c=False*)
Bases: intermol.forces.cosine_angle_type.CosineAngleType

http://lammps.sandia.gov/doc/angle_cosine.html

class intermol.forces.cosine_angle_type.**CosineAngleType**(**args*, ***kwds*)
Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

intermol.forces.cosine_squared_angle_type module

class intermol.forces.cosine_squared_angle_type.**CosineSquaredAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *c=False*)
Bases: intermol.forces.cosine_squared_angle_type.CosineSquaredAngleType

stub documentation

class intermol.forces.cosine_squared_angle_type.**CosineSquaredAngleType**(**args*, ***kwds*)
Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

theta

intermol.forces.cross_bond_angle_angle_type module

class intermol.forces.cross_bond_angle_angle_type.**CrossBondAngleAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *r1=Quantity(value=0.0, unit=nanometer)*, *r2=Quantity(value=0.0, unit=nanometer)*, *r3=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mo*, *c=False*)

Bases: intermol.forces.cross_bond_angle_angle_type.CrossBondAngleAngleType

stub documentation

class intermol.forces.cross_bond_angle_angle_type.**CrossBondAngleAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

r1

r2

r3

intermol.forces.cross_bond_bond_angle_type module

class intermol.forces.cross_bond_bond_angle_type.**CrossBondBondAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *r1=Quantity(value=0.0, unit=nanometer)*, *r2=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole)*, *c=False*)

Bases: intermol.forces.cross_bond_bond_angle_type.CrossBondBondAngleType

stub documentation

class intermol.forces.cross_bond_bond_angle_type.**CrossBondBondAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

r1

r2

intermol.forces.cubic_bond_type module

class intermol.forces.cubic_bond_type.**CubicBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *C2=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *C3=Quantity(value=0.0, unit=kilojoule/(nanometer**3*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.cubic_bond_type.CubicBondType

stub documentation

class intermol.forces.cubic_bond_type.**CubicBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

C2

C3

c

length

order

intermol.forces.fene_bond_type module

class intermol.forces.fene_bond_type.**FeneBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *kb=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.fene_bond_type.FeneBondType

stub documentation

class intermol.forces.fene_bond_type.**FeneBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

kb

length

order

intermol.forces.fene_expandable_bond_type module

class intermol.forces.fene_expandable_bond_type.**FeneExpandableBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *length=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *sigma=Quantity(value=0.0, unit=nanometer)*, *delta=Quantity(value=0.0, unit=nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.fene_expandable_bond_type.FeneExpandableBondType

stub documentation

class intermol.forces.fene_expandable_bond_type.**FeneExpandableBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

delta

epsilon

k

length

order

sigma

intermol.forces.forcedata module

intermol.forces.forcefunctions module

intermol.forces.forcefunctions.**build_paramlist**(*program*)
Create a paramlist specific for a given program.

intermol.forces.forcefunctions.**build_unitvars**(*program*, *paramlist*, *dumself=None*)
Takes a string program name (one of the supported programs), and a 'self' object it looks like the keyword is not being used, but it is used in the line eval(unit). The test name 'dumself' needs to match what is in the force data arrays. Currently only used for lammps.

intermol.forces.forcefunctions.**capifyname**(*forcename*)
Return name of the class in camelCase.

intermol.forces.forcefunctions.**create_kwd_dict**(*unitvars*, *paramlist*, *force_type_object*, *values*, *optvalues=None*)

intermol.forces.forcefunctions.**create_kwds_from_entries**(*unitvars*, *paramlist*, *entries*, *force_type*, *offset=0*)

Create a keyword dictionary given an array of information from a file format

requires the master set of units, the master set of parameter lists, an object (either a force_class or force_type), the list of information to be converted into a keyword, and an offset.

Parameters **offset** (*int*) – how far over from the first entry we translate

intermol.forces.forcefunctions.**get_parameter_kwds_from_force**(*force*, *forceparams*, *paramlist*)

intermol.forces.forcefunctions.**get_parameter_list_from_force**(*force*, *paramlist*)

Create a function that returns the paramters of a function type.

First, we need make some additions to the parameter list dictionary, which we do once when the forcedata script is imported. Useful to put the forces here as well. We won't make this a function for now since it's needed in this module.

intermol.forces.forcefunctions.**get_parameter_list_from_kwds**(*force*, *kwds*, *paramlist*)

intermol.forces.forcefunctions.**optforceparams**(*force_type*, *forcetype_object=None*)

Return the dictionary of optional paramters of an abstract force type.

If no object is given, we fill with blanks.

intermol.forces.forcefunctions.**optparamkeylookup**(*force_type*)

Given a force_type object, determine the key associated with the optional parameters.

intermol.forces.forcefunctions.**optparamlookup**(*force_type_object*, *object_default=False*)

A wrapper for optforceparams that takes a force_type object and returns the optional parameter dictionary.

intermol.forces.forcefunctions.**specify**(*program_units*, *unitset*, *dumself=None*, *shouldEval=True*)

Takes the dict of units, and a set of dimensions and replaces the dimensions with the appropriate units.

intermol.forces.four_fdn_virtual_type module

class intermol.forces.four_fdn_virtual_type.**FourFdnVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *atom5*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *bondingtype5=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *c=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.four_fdn_virtual_type.FourFdnVirtualType

stub documentation

class intermol.forces.four_fdn_virtual_type.**FourFdnVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_4_virtual_type.Abstract4VirtualType

a

b

c

placeholder

intermol.forces.fourier_dihedral_type module

class intermol.forces.fourier_dihedral_type.**FourierDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *c1=Quantity(value=0.0, unit=kilojoule/mole)*, *c2=Quantity(value=0.0, unit=kilojoule/mole)*, *c3=Quantity(value=0.0, unit=kilojoule/mole)*, *c4=Quantity(value=0.0, unit=kilojoule/mole)*, *c5=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.fourier_dihedral_type.FourierDihedralType

stub documentation

class intermol.forces.fourier_dihedral_type.**FourierDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

c1

c2

c3

c4

c5

improper

intermol.forces.g96_bond_type module

class intermol.forces.g96_bond_type.**G96Bond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**4*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.g96_bond_type.G96BondType

stub documentation

class intermol.forces.g96_bond_type.**G96BondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.harmonic_angle_type module

class intermol.forces.harmonic_angle_type.**HarmonicAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *c=False*)

Bases: intermol.forces.harmonic_angle_type.HarmonicAngleType

stub documentation

class intermol.forces.harmonic_angle_type.**HarmonicAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

theta

intermol.forces.harmonic_bond_type module

class intermol.forces.harmonic_bond_type.**HarmonicBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.harmonic_bond_type.HarmonicBondType

stub documentation

class intermol.forces.harmonic_bond_type.**HarmonicBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.harmonic_potential_bond_type module

class intermol.forces.harmonic_potential_bond_type.**HarmonicPotentialBond**(*atom1*, *atom2*, *bonding_type1=None*, *bonding_type2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**2*, *order=1*, *c=False*)

Bases: intermol.forces.harmonic_potential_bond_type.HarmonicPotentialBondType

stub documentation

class intermol.forces.harmonic_potential_bond_type.**HarmonicPotentialBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

k

length

order

intermol.forces.improper_harmonic_dihedral_type module

class intermol.forces.improper_harmonic_dihedral_type.**ImproperHarmonicDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *xi=Quantity(value=0 unit=degree)*, *k=Quantity(value=0 unit=kilojoule/(mole*, *improper=False*)

Bases: intermol.forces.improper_harmonic_dihedral_type.ImproperHarmonicDihedralType

stub documentation

class intermol.forces.improper_harmonic_dihedral_type.**ImproperHarmonicDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

improper

k

xi

intermol.forces.lj_c_nonbonded_type module

class intermol.forces.lj_c_nonbonded_type.**LjCNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *type=False*)

Bases: intermol.forces.lj_c_nonbonded_type.LjCNonbondedType

stub documentation

class intermol.forces.lj_c_nonbonded_type.**LjCNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

C12

C6

type

intermol.forces.lj_c_pair_type module

class intermol.forces.lj_c_pair_type.**LjCPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_c_pair_type.LjCPairType

stub documentation

class intermol.forces.lj_c_pair_type.**LjCPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

C12

C6

long

scaleLJ

scaleQQ

intermol.forces.lj_default_pair_type module

class intermol.forces.lj_default_pair_type.**LjDefaultPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_default_pair_type.LjDefaultPairType

stub documentation

class intermol.forces.lj_default_pair_type.**LjDefaultPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

long

scaleLJ

scaleQQ

intermol.forces.lj_sigeps_nonbonded_type module

class intermol.forces.lj_sigeps_nonbonded_type.**LjSigepsNonbonded**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *type=False*)

Bases: intermol.forces.lj_sigeps_nonbonded_type.LjSigepsNonbondedType

stub documentation

class intermol.forces.lj_sigeps_nonbonded_type.**LjSigepsNonbondedType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_nonbonded_type.AbstractNonbondedType

epsilon

sigma

type

intermol.forces.lj_sigeps_pair_type module

class intermol.forces.lj_sigeps_pair_type.**LjSigepsPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.lj_sigeps_pair_type.LjSigepsPairType

stub documentation

class intermol.forces.lj_sigeps_pair_type.**LjSigepsPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

epsilon

long

scaleLJ

scaleQQ

sigma

intermol.forces.ljq_c_pair_type module

class intermol.forces.ljq_c_pair_type.**LjqCPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *qi=Quantity(value=0.0, unit=elementary charge)*, *qj=Quantity(value=0.0, unit=elementary charge)*, *C6=Quantity(value=0.0, unit=nanometer**6*kilojoule/mole)*, *C12=Quantity(value=0.0, unit=nanometer**12*kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_c_pair_type.LjqCPairType

stub documentation

class intermol.forces.ljq_c_pair_type.**LjqCPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

C12

C6

long

qi

qj

scaleLJ

scaleQQ

intermol.forces.ljq_default_pair_type module

class intermol.forces.ljq_default_pair_type.**LjqDefaultPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_default_pair_type.LjqDefaultPairType

stub documentation

class intermol.forces.ljq_default_pair_type.**LjqDefaultPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

long

scaleLJ

scaleQQ

intermol.forces.ljq_sigeps_pair_type module

class intermol.forces.ljq_sigeps_pair_type.**LjqSigepsPair**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *qi=Quantity(value=0.0, unit=elementary charge)*, *qj=Quantity(value=0.0, unit=elementary charge)*, *sigma=Quantity(value=0.0, unit=nanometer)*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *scaleLJ=None*, *scaleQQ=None*, *long=False*)

Bases: intermol.forces.ljq_sigeps_pair_type.LjqSigepsPairType

stub documentation

class intermol.forces.ljq_sigeps_pair_type.**LjqSigepsPairType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_pair_type.AbstractPairType

epsilon

long

qi

qj

scaleLJ

scaleQQ

sigma

intermol.forces.make_forces module

intermol.forces.morse_bond_type module

class intermol.forces.morse_bond_type.**MorseBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *D=Quantity(value=0.0, unit=kilojoule/mole)*, *beta=Quantity(value=0.0, unit=/nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.morse_bond_type.MorseBondType

stub documentation

class intermol.forces.morse_bond_type.**MorseBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

D

beta

c

length

order

intermol.forces.nonlinear_bond_type module

class intermol.forces.nonlinear_bond_type.**NonlinearBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *epsilon=Quantity(value=0.0, unit=kilojoule/mole)*, *r0=Quantity(value=0.0, unit=nanometer)*, *lamda=Quantity(value=0.0, unit=nanometer)*, *order=1*, *c=False*)

Bases: intermol.forces.nonlinear_bond_type.NonlinearBondType

http://lammps.sandia.gov/doc/bond_nonlinear.html

class intermol.forces.nonlinear_bond_type.**NonlinearBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

c

epsilon

lamda

order

r0

intermol.forces.proper_periodic_dihedral_type module

class intermol.forces.proper_periodic_dihedral_type.**ProperPeriodicDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bonding-type1=None*, *bonding-type2=None*, *bonding-type3=None*, *bonding-type4=None*, *phi=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/mole)*, *multiplicity=Quantity(value=0.0, unit=dimensionless)*, *weight=Quantity(value=0.0, unit=dimensionless)*, *improper=False*)

Bases: intermol.forces.proper_periodic_dihedral_type.ProperPeriodicDihedralType

stub documentation

class intermol.forces.proper_periodic_dihedral_type.**ProperPeriodicDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

improper

k

multiplicity

phi

weight

intermol.forces.quartic_angle_type module

class intermol.forces.quartic_angle_type.**QuarticAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *C0=Quantity(value=0.0, unit=kilojoule/mole)*, *C1=Quantity(value=0.0, unit=kilojoule/(mole*radian))*, *C2=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *C3=Quantity(value=0.0, unit=kilojoule/(mole*radian**3))*, *C4=Quantity(value=0.0, unit=kilojoule/(mole*radian**4))*, *c=False*)

Bases: intermol.forces.quartic_angle_type.QuarticAngleType

stub documentation

class intermol.forces.quartic_angle_type.**QuarticAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

C0

C1

C2

C3

C4

c

theta

intermol.forces.quartic_bond_type module

class intermol.forces.quartic_bond_type.**QuarticBond**(*atom1*, *atom2*, *bondingtype1=None*, *bondingtype2=None*, *length=Quantity(value=0.0, unit=nanometer)*, *C2=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *C3=Quantity(value=0.0, unit=kilojoule/(nanometer**3*mole))*, *C4=Quantity(value=0.0, unit=kilojoule/(nanometer**4*mole))*, *order=1*, *c=False*)

Bases: intermol.forces.quartic_bond_type.QuarticBondType

stub documentation

class intermol.forces.quartic_bond_type.**QuarticBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

C2

C3

C4

c

length

order

intermol.forces.quartic_breakable_bond_type module

class intermol.forces.quartic_breakable_bond_type.**QuarticBreakableBond**(*atom1*, *atom2*, *bonding_type1=None*, *bonding_type2=None*, *k=Quantity(value=0.0, unit=kilojoule/(nanometer**4*r*, *B1=Quantity(value=0.0, unit=nanometer)*, *B2=Quantity(value=0.0, unit=nanometer)*, *Rc=Quantity(value=0.0, unit=nanometer)*, *U0=Quantity(value=0.0, unit=kilojoule/mole)*, *order=1*, *c=False*)

Bases: intermol.forces.quartic_breakable_bond_type.QuarticBreakableBondType

http://lammps.sandia.gov/doc/bond_quartic.html

class intermol.forces.quartic_breakable_bond_type.**QuarticBreakableBondType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_bond_type.AbstractBondType

B1

B2

Rc

U0

c

k

order

intermol.forces.rb_dihedral_type module

class intermol.forces.rb_dihedral_type.**RbDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *C0=Quantity(value=0.0, unit=kilojoule/mole)*, *C1=Quantity(value=0.0, unit=kilojoule/mole)*, *C2=Quantity(value=0.0, unit=kilojoule/mole)*, *C3=Quantity(value=0.0, unit=kilojoule/mole)*, *C4=Quantity(value=0.0, unit=kilojoule/mole)*, *C5=Quantity(value=0.0, unit=kilojoule/mole)*, *C6=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.rb_dihedral_type.RbDihedralType

stub documentation

class intermol.forces.rb_dihedral_type.**RbDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

C0

C1

C2

C3

C4

C5

C6

improper

intermol.forces.settles module

class intermol.forces.settles.**Settles**(**args*, ***kwds*)

Bases: intermol.forces.abstract_type.AbstractType

intermol.forces.three_fad_virtual_type module

class intermol.forces.three_fad_virtual_type.**ThreeFadVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *theta=Quantity(value=0.0, unit=degree)*, *d=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_fad_virtual_type.ThreeFadVirtualType

stub documentation

class intermol.forces.three_fad_virtual_type.**ThreeFadVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

d

placeholder

theta

intermol.forces.three_fd_virtual_type module

class intermol.forces.three_fd_virtual_type.**ThreeFdVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *d=Quantity(value=0.0, unit=nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_fd_virtual_type.ThreeFdVirtualType

stub documentation

class intermol.forces.three_fd_virtual_type.**ThreeFdVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

d

placeholder

intermol.forces.three_linear_virtual_type module

class intermol.forces.three_linear_virtual_type.**ThreeLinearVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *placeholder=False*)

Bases: intermol.forces.three_linear_virtual_type.ThreeLinearVirtualType

stub documentation

class intermol.forces.three_linear_virtual_type.**ThreeLinearVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

b

placeholder

intermol.forces.three_out_virtual_type module

class intermol.forces.three_out_virtual_type.**ThreeOutVirtual**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *b=Quantity(value=0.0, unit=dimensionless)*, *c=Quantity(value=0.0, unit=/nanometer)*, *placeholder=False*)

Bases: intermol.forces.three_out_virtual_type.ThreeOutVirtualType

stub documentation

class intermol.forces.three_out_virtual_type.**ThreeOutVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_3_virtual_type.Abstract3VirtualType

a

b

c

placeholder

intermol.forces.trig_dihedral_type module

class intermol.forces.trig_dihedral_type.**TrigDihedral**(*atom1*, *atom2*, *atom3*, *atom4*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *bondingtype4=None*, *phi=Quantity(value=0.0, unit=degree)*, *fc0=Quantity(value=0.0, unit=kilojoule/mole)*, *fc1=Quantity(value=0.0, unit=kilojoule/mole)*, *fc2=Quantity(value=0.0, unit=kilojoule/mole)*, *fc3=Quantity(value=0.0, unit=kilojoule/mole)*, *fc4=Quantity(value=0.0, unit=kilojoule/mole)*, *fc5=Quantity(value=0.0, unit=kilojoule/mole)*, *fc6=Quantity(value=0.0, unit=kilojoule/mole)*, *improper=False*)

Bases: intermol.forces.trig_dihedral_type.TrigDihedralType

stub documentation

class intermol.forces.trig_dihedral_type.**TrigDihedralType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_dihedral_type.AbstractDihedralType

fc0

fc1

fc2

fc3

fc4

fc5

fc6

improper

phi

intermol.forces.two_virtual_type module

class intermol.forces.two_virtual_type.**TwoVirtual**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *a=Quantity(value=0.0, unit=dimensionless)*, *placeholder=False*)

Bases: intermol.forces.two_virtual_type.TwoVirtualType

stub documentation

class intermol.forces.two_virtual_type.**TwoVirtualType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_2_virtual_type.Abstract2VirtualType

a

placeholder

intermol.forces.urey_bradley_angle_type module

class intermol.forces.urey_bradley_angle_type.**UreyBradleyAngle**(*atom1*, *atom2*, *atom3*, *bondingtype1=None*, *bondingtype2=None*, *bondingtype3=None*, *theta=Quantity(value=0.0, unit=degree)*, *k=Quantity(value=0.0, unit=kilojoule/(mole*radian**2))*, *r=Quantity(value=0.0, unit=nanometer)*, *kUB=Quantity(value=0.0, unit=kilojoule/(nanometer**2*mole))*, *c=False*)

Bases: intermol.forces.urey_bradley_angle_type.UreyBradleyAngleType

stub documentation

class intermol.forces.urey_bradley_angle_type.**UreyBradleyAngleType**(**args*, ***kwds*)

Bases: intermol.forces.abstract_angle_type.AbstractAngleType

c

k

kUB

r

theta

3.4.2 Module contents

CHAPTER 4

License

InterMol is licensed under the MIT license.

CHAPTER 5

Indices and tables

- *genindex*
- *modindex*
- *search*

i

Symbols

A

B

C

D

E

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X