pysimm

Release 1.0

Mike Fortunato

Jul 12, 2021

CONTENTS:

pysimm is a python package designed to facilitate structure generation, simulation, and modification of molecular systems by providing a collection of simulation tools and smooth integration with highly optimized third party software. Abstraction layers enable a standardized methodology to assign various force field models to molecular systems and perform simple simulations.

To read more, see our publication in SoftwareX.

CHAPTER

ONE

API REFERENCE

This page contains auto-generated API reference documentation[1].

1.1 pysimm

1.1.1 Subpackages

pysimm.apps

Submodules

pysimm.apps.equilibrate

Module Contents

Functions

equil(s, **kwargs)	pysimm.apps.equilibrate.equil

Attributes

rappture
rappture

pysimm.apps.equilibrate.**rappture** = **True**

pysimm.apps.equilibrate.**rappture** = **False**

pysimm.apps.equilibrate.**equil**(*s*, ***kwargs*)

pysimm.apps.equilibrate.equil

Runs a 21-step compression/decompression equilibration algorithm

Parameters

[1] Created with sphinx-autoapi

- **s** – *System* object
- **tmax** – maximum temperature during equilibration
- **pmax** – maximum pressure during equilibration
- **tfinal** – desired final temperature of final system
- **pfinal** – desired final pressure of final system
- **np** – number of processors to use during equilibration simulations
- **p_steps** – list of pressures to use during equilibration (must match length of length_list)
- **length_list** – list of simulation durations to use during equilibration (must match length of p_steps)

Returns None

`pysimm.apps.mc_md`

Module Contents

Functions

mc_md(gas_sst, fixed_sst=None, mcmd_niter=None, sim_folder=None, mc_props=None, md_props=None, **kwargs)	pysimm.apps.mc_md

`pysimm.apps.mc_md.`**`mc_md`**(*gas_sst*, *fixed_sst=None*, *mcmd_niter=None*, *sim_folder=None*, *mc_props=None*, *md_props=None*, ***kwargs*)

pysimm.apps.mc_md

Performs the iterative hybrid Monte-Carlo/Molecular Dynamics (MC/MD) simulations using *lmps* for MD and *cassandra* for MC

Parameters

- **gas_sst** (list of *System*) – list items describe a different molecule to be inserted by MC
- **fixed_sst** (*System*) – fixed during th MC steps group of atoms (default: None)

Keyword Arguments

- **mcmd_niter** (*int*) – number of MC-MD iterations (default: 10)
- **sim_folder** (*str*) – relative path to the folder with all simulation files (default: 'results')
- **mc_props** (*dictionary*) – description of all MC properties needed for simulations (see *GCMC* and `props` for details)
- **md_props** (*dictionary*) – description of all Molecular Dynamics settings needed for simulations (see *Simulation* and *MolecularDynamics* for details)

Returns Final state of the simulated system

Return type *System*

`pysimm.apps.polymatic`

Module Contents

Functions

pack(script, file_in, nrep, boxl, file_out)	pysimm.apps.polymatic.pack
polymatic(script, file_in, file_out)	pysimm.apps.polymatic.polymatic
run(settings)	pysimm.apps.polymatic.run
lmps_min(s, name, settings)	pysimm.apps.polymatic.lmps_min
lmps_step_md(s, bonds, attempt, settings)	pysimm.apps.polymatic.lmps_step_md
lmps_cycle_nvt_md(s, bonds, settings)	pysimm.apps.polymatic.lmps_cycle_nvt_md
lmps_cycle_npt_md(s, bonds, settings)	pysimm.apps.polymatic.lmps_cycle_npt_md

Attributes

rappture	
rappture	

`pysimm.apps.polymatic.`**`rappture = True`**

`pysimm.apps.polymatic.`**`rappture = False`**

`pysimm.apps.polymatic.`**`pack`**`(`*script*, *file_in*, *nrep*, *boxl*, *file_out*`)`
pysimm.apps.polymatic.pack

Calls Polymatic random packing code

Parameters

- **`script`** – name of packing script
- **`file_in`** – list of file names of reference molecules to pack
- **`nrep`** – list of number of monomers for each reference molecule
- **`boxl`** – length of one dimension of simulation box for random packing
- **`file_out`** – name of output file (packed system)

Returns output from perl code

`pysimm.apps.polymatic.`**`polymatic`**`(`*script*, *file_in*, *file_out*`)`
pysimm.apps.polymatic.polymatic

Calls Polymatic code. polym.in and types.txt are assumed to exist.

Parameters

- **`script`** – name of Polymatic script
- **`file_in`** – initial system file name
- **`file_out`** – final system file name

Returns output from perl code

`pysimm.apps.polymatic.`**`run`**`(`*settings*`)`

pysimm.apps.polymatic.run

Runs Polymatic algorithm.

Parameters **`settings`** – object containing Polymatic settings

Returns (True/False, *`System`*)

`pysimm.apps.polymatic.`**`lmps_min`**`(`*s*, *name*, *settings*`)`

pysimm.apps.polymatic.lmps_min

Runs LAMMPS minimization for the Polymatic algorithm.

Parameters

- **`s`** – *`System`* to minimize
- **`name`** – name of simulation
- **`settings`** – object containing Polymatic settings

Returns result from `minimize()`

`pysimm.apps.polymatic.`**`lmps_step_md`**`(`*s*, *bonds*, *attempt*, *settings*`)`

pysimm.apps.polymatic.lmps_step_md

Runs LAMMPS step md for the Polymatic algorithm.

Parameters

- **`s`** – *`System`* to minimize
- **`bonds`** – number of bond to be made
- **`attempt`** – number of bonding attempt
- **`settings`** – object containing Polymatic settings

Returns result from `md()`

`pysimm.apps.polymatic.`**`lmps_cycle_nvt_md`**`(`*s*, *bonds*, *settings*`)`

pysimm.apps.polymatic.lmps_cycle_nvt_md

Runs LAMMPS nvt cycle md for the Polymatic algorithm.

Parameters

- **`s`** – *`System`* to minimize
- **`bonds`** – number of bond to be made
- **`settings`** – object containing Polymatic settings

Returns result from `md()`

`pysimm.apps.polymatic.`**`lmps_cycle_npt_md`**`(`*s*, *bonds*, *settings*`)`

pysimm.apps.polymatic.lmps_cycle_npt_md

Runs LAMMPS npt cycle md for the Polymatic algorithm.

Parameters

- **`s`** – *`System`* to minimize
- **`bonds`** – number of bond to be made
- **`settings`** – object containing Polymatic settings

Returns result from lmps.md

pysimm.apps.poreblazer

Module Contents

Functions

psd(s, **kwargs)	pysimm.apps.poreblazer.psd
surface(s, **kwargs)	pysimm.apps.poreblazer.surface
pore(s, **kwargs)	pysimm.apps.poreblazer.pore
void(s, **kwargs)	pysimm.apps.poreblazer.void
psd3(s, **kwargs)	pysimm.apps.poreblazer.psd3

Attributes

boltzmann_kcal

pysimm.apps.poreblazer.**boltzmann_kcal = 0.001987204**

pysimm.apps.poreblazer.**psd**(*s*, ***kwargs*)

pysimm.apps.poreblazer.psd

Perform pore size distribution calculation using PoreBlazer v2.0

Parameters

- **atoms** – file name to contain ff parameters (ff.atoms)
- **data** – file name to write xyz file (data.xyz)
- **angles** – angles of simlation box (90.0 90.0 90.0)
- **insertions** – number of insertions for calculation (500)
- **min_probe** – minimum probe size (1.0)
- **probe_dr** – step size to increase probe size (0.2)
- **max_probe** – maximum probe size: 25
- **psd_save** – T/F to save psd points (F)
- **psd_range** – range in which to save psd points (2.5,3.8)
- **exec_path** – path to poreblazer psd executable (psd.exe)
- **gen_files** – if True, only generate input do not execute (None)

Returns None

pysimm.apps.poreblazer.**surface**(*s*, ***kwargs*)

pysimm.apps.poreblazer.surface

Perform accessible surface area calculation using PoreBlazer v2.0

Parameters

- **atoms** – file name to contain ff parameters (ff.atoms)

- `data` – file name to write xyz file (data.xyz)
- `angles` – angles of simlation box (90.0 90.0 90.0)
- `insertions` – number of insertions for calculation (1000)
- `probe` – probe size (3.681)
- `probe_type` – type of probe (hs)
- `vis` – True to save visual (F)
- `exec_path` – path to poreblazer surface executable (surface.exe)

Returns None

`pysimm.apps.poreblazer.pore`(*s*, ***kwargs*)

pysimm.apps.poreblazer.pore

Perform pore volume calculation using PoreBlazer v2.0

Parameters

- `atoms` – file name to contain ff parameters (ff.atoms)
- `data` – file name to write xyz file (data.xyz)
- `angles` – angles of simlation box (90.0 90.0 90.0)
- `insertions` – number of insertions for calculation (1000)
- `temp` – temperature at which to perform simulation (300)
- `pore_probe` – sigma, epsilon, cutoff parameters for probe (2.58, 10.22, 12.8)
- `exec_path` – path to poreblazer pore executable (pore_he.exe)

Returns None

`pysimm.apps.poreblazer.void`(*s*, ***kwargs*)

pysimm.apps.poreblazer.void

Perform pore volume calculation using PoreBlazer v2.0 assuming a probe size of 0 to calculate void volume

Parameters

- `atoms` – file name to contain ff parameters (ff.atoms)
- `data` – file name to write xyz file (data.xyz)
- `angles` – angles of simlation box (90.0 90.0 90.0)
- `insertions` – number of insertions for calculation (1000)
- `temp` – temperature at which to perform simulation (300)
- `pore_probe` – sigma, epsilon, cutoff parameters for probe (0.00, 10.22, 12.8)
- `exec_path` – path to poreblazer pore executable (pore_he.exe)

Returns None

`pysimm.apps.poreblazer.psd3`(*s*, ***kwargs*)

pysimm.apps.poreblazer.psd3

Perform combined pore volume, surface area, pore accessibility calculation using PoreBlazer v3.0 or later. For more detailed description of input parameters please check the Poreblazer GitHub page: https://github.com/SarkisovGroup/PoreBlazer

Parameters

- **atoms** – file name to contain ff parameters (ff.atoms)
- **data** – file name to write xyz file (data.xyz)
- **angles** – angles of simulation box (90.0 90.0 90.0)
- **he_params** – dictionary containing parameters for the helium atom probe: he_params['sigma']: LJ sigma parameter [in Å] (2.58) he_params['eps']: LJ epsilon parameter [in K] (10.22) he_params['temp']: temperature [in K], required for the Helium porosimetry (300) he_params['cutoff']: cut-off distance [in Å], required for the Helium porosimetry (12.0)
- **probe** – nitrogen atom probe size [in Å] (3.314)
- **insertions** – number of trials per atom for the surface area calculations (500)
- **probe_dr** – linear size of a cell of the uniform grid [in Å] (0.2)
- **exec_path** – full (relative or absolute) path to Poreblazer executable (poreblazer.exe)

Returns 0 if the Poreblazer finished successfully, and 1 otherwise

Return type Exit status

pysimm.apps.random_walk

Module Contents

Functions

displ_next_unit_default(m, s)	pysimm.apps.random_walk.displ_next_unit_default
find_last_backbone_vector(s, m)	pysimm.apps.random_walk.find_last_backbone_vector
copolymer(m, nmon, s_=None, **kwargs)	pysimm.apps.random_walk.copolymer
random_walk(m, nmon, s_=None, **kwargs)	pysimm.apps.random_walk.random_walk
find_last_tail_vector(s)	pysimm.apps.random_walk.find_last_tail_vector
rot_mat_about_axis(v, theta)	pysimm.apps.random_walk.rot_mat_about_axis
define_plane(a1, a2, a3)	pysimm.apps.random_walk.define_plane
reflect_coords_thru_plane(atom, plane)	pysimm.apps.random_walk.reflect_coords_thru_plane
scale_monomer(atom, origin, scale)	pysimm.apps.random_walk.scale_monomer
redo_monomer_insertion(s, m, i)	pysimm.apps.random_walk.redo_monomer_insertion
constrained_opt(s, m, active)	pysimm.apps.random_walk.constrained_opt
random_walk_tacticity(m, nmon, s_=None, **kwargs)	pysimm.apps.random_walk.random_walk_tacticity
__check_tags__(m, **kwargs)	pysimm.apps.random_walk.__check_tags__
check_tacticity(s, char_idxs, mon_len)	pysimm.apps.random_walk.check_tacticity

pysimm.apps.random_walk.**displ_next_unit_default**(*m*, *s*)

pysimm.apps.random_walk.displ_next_unit_default

Default implementation of displacement of next repetitive unit in random walk method for a polymer growth

Parameters

- **m** – *System* object – updated
- **s** – *ItemContainer* of *~pysimm.system.Particle* objects

Returns general placeholder for connectivity order of linker atoms; default implementation assumes work with polymers with a single linkage bond, thus no connectivity order is needed

Return type empty list

`pysimm.apps.random_walk.`**`find_last_backbone_vector`**`(`*s*, *m*`)`

pysimm.apps.random_walk.find_last_backbone_vector

Finds vector between backbone atoms in terminal monomer. Requires current system s, and reference monomer m.

Parameters

- **`s`** – *`System`* object
- **`m`** – *`System`* object

Returns list of vector components

`pysimm.apps.random_walk.`**`copolymer`**`(`*m*, *nmon*, *s_=None*, ***kwargs*`)`

pysimm.apps.random_walk.copolymer

Builds copolymer using random walk methodology using pattern

Parameters

- **`m`** – list of reference monomer :class:`~pysimm.system.System`s
- **`nmon`** – total number of monomers to add to chain
- **`s`** – *`System`* in which to build polymer chain (None)
- **`settings`** – dictionary of simulation settings
- **`density`** – density at which to build polymer (0.3)
- **`forcefield`** – *`Forcefield`* object to acquire new force field parameters
- **`capped`** – True/False if monomers are capped
- **`unwrap`** – True to unwrap final system
- **`traj`** – True to build xyz trajectory of polymer growth (True)
- **`pattern`** – list of pattern for monomer repeat units, should match length of m ([1 for _ in range(len(m))])
- **`limit`** – during MD, limit atomic displacement by this max value (LAMMPS ONLY)
- **`sim`** – *`Simulation`* object for relaxation between polymer growth

Returns new copolymer *`System`*

`pysimm.apps.random_walk.`**`random_walk`**`(`*m*, *nmon*, *s_=None*, ***kwargs*`)`

pysimm.apps.random_walk.random_walk

Builds homopolymer using random walk methodology

Parameters

- **`m`** – reference monomer *`System`*
- **`nmon`** – total number of monomers to add to chain
- **`s`** – *`System`* in which to build polymer chain (None)
- **`extra_bonds`** – EXPERMINTAL, True if making ladder backbone polymer
- **`geometry_rule`** – a pointer to a method that orients series of atoms of the next repetitive unit in random run

- **`settings`** – dictionary of simulation settings
- **`density`** – density at which to build polymer (0.3)
- **`forcefield`** – *`Forcefield`* object to acquire new force field parameters
- **`capped`** – True/False if monomers are capped
- **`unwrap`** – True to unwrap final system
- **`traj`** – True to build xyz trajectory of polymer growth (True)
- **`limit`** – during MD, limit atomic displacement by this max value (LAMMPS ONLY)
- **`sim`** – *`Simulation`* object for relaxation between polymer growth
- **`debug`** – Boolean; print extra-output

Returns new polymer *`System`*

`pysimm.apps.random_walk.`**`find_last_tail_vector`**`(`*s*`)`

pysimm.apps.random_walk.find_last_tail_vector Finds vector defined by bond in the system between the tail atom and its capping atom. Requires list of particles s that formed a monomer connected on previous step of the polymerisation.

Parameters **`s`** – ItemContainer of *`Particle`* objects

Returns list of vector components

`pysimm.apps.random_walk.`**`rot_mat_about_axis`**`(`*v*, *theta*`)`

pysimm.apps.random_walk.rot_mat_about_axis This function returns the matrix that represents a rotation about vector v by theta degrees. Used for isotactic insertions of monomers

Parameters

- **`v`** – vector about which to rotate
- **`theta`** – degrees to rotate

Returns matrix representation of rotation

`pysimm.apps.random_walk.`**`define_plane`**`(`*a1*, *a2*, *a3*`)`

pysimm.apps.random_walk.define_plane This function returns the mathematical constants defining a plane containing three input particles

Parameters

- **`a1`** – three atoms or particles
- **`a2`** – three atoms or particles
- **`a3`** – three atoms or particles

Returns np.array containing a,b,c,d that define the plane a*x + b*y + c*z + d = 0 that contains the input particles

`pysimm.apps.random_walk.`**`reflect_coords_thru_plane`**`(`*atom*, *plane*`)`

pysimm.apps.random_walk.reflect_coords_thru_plane This function reflects an atom through a plane, and is used for implementing syndiotactic insertions of monomers

Parameters

- **`atom`** – either an atom or an array containing x,y,z coordinates for an atom, to be reflected through the plane
- **`plane`** – np.array containing a,b,c,d that define a plane, a*x + b*y + c*z + d = 0

Returns new coordinates after reflection through plane `pysimm.apps.random_walk.`**`scale_monomer`**`(`*atom*, *origin*, *scale*`)`

pysimm.apps.random_walk.scale_monomer This function scales the atom–origin vector. It is used by redo_monomer_insertion to scale the last monomer relative to its attachment point to the polymer chain

Parameters

- **`atom`** – either an atom or an array containing x,y,z coordinates for an atom, to be scaled relative to the origin
- **`origin`** – either an atom or an array containing x,y,z coordinates for where the "atom" argument should be scaled to
- **`scale`** – the factor by which the atom–origin vector should be scaled.

Returns scaled atom–origin vector

`pysimm.apps.random_walk.`**`redo_monomer_insertion`**`(`*s*, *m*, *i*`)`

pysimm.apps.random_walk.redo_monomer_insertion This function is called by random_walk_tacticity if the latest capped monomer insertion resulted in hardcore overlaps. 1) The hardcore overlap is resolved by shrinking the last monomer by a factor of 0.8, iteratively, until there are no more hardcore overlaps. 2) Then the shrunken last monomer is frozen while the rest of the polymer chain is optimized, and the last monomer is scaled in size by 1.05 3) Cycles of contrainedOptimization and regrowth are alternated until a reasonable structure is obtained

Parameters

- **`s`** – *`System`* is a polymer chain in which the last monomer insertion has generated a hardcore overlap
- **`m`** – reference monomer *`System`*. Must be a capped monomer, with headCap and tail_cap as the first and last atoms in the .mol file.
- **`i`** – number of the offending monomer, used for labelling diagnostic .xyz output files

Returns nothing; all changes to the polymer chain are written to the argument **s_**

`pysimm.apps.random_walk.`**`constrained_opt`**`(`*s*, *m*, *active*`)`

pysimm.apps.random_walk.constrained_opt This function is called by redo_monomer_insertion and optimizes polymer chain s while keeping the last monomer fixed.

Parameters

- **`s`** – *`System`* is a polymer chain in which the last monomer insertion has generated a hardcore overlap
- **`m`** – reference monomer *`System`*. Must be a capped monomer, with headCap and tail_cap as the first and last atoms in the .mol file.

Returns nothing; all changes to the polymer chain are written to the argument **s_**

`pysimm.apps.random_walk.`**`random_walk_tacticity`**`(`*m*, *nmon*, *s_=None*, ***kwargs*`)`

pysimm.apps.random_walk.random_walk_tacticity Builds homopolymer with controllable tacticity from capped monomer structure

Parameters

- **`m`** – reference monomer *`System`*. Must be a capped monomer, with headCap and tail_cap
- **`file.`** (*`as the first and last atoms in the .mol`*) –
- **`nmon`** – total number of monomers to add to chain
- **`s`** – *`System`* in which to build polymer chain (None)
- **`extra_bonds`** – EXPERMINTAL, True if making ladder backbone polymer
- **`settings`** – dictionary of simulation settings

- **density** – density at which to build polymer (0.3)
- **forcefield** – *Forcefield* object to acquire new force field parameters
- **unwrap** – True to unwrap final system
- **debug** – Boolean; print extra-output (False)
- **traj** – True to build xyz trajectory of polymer growth (True)
- **limit** – during MD, limit atomic displacement by this max value (LAMMPS ONLY)
- **sim** – *Simulation* object for relaxation between polymer growth
- **tacticity** – float between 0 and 1. 1 = 100% isotactic insertions 0 = 100% syndiotactic insertions 0.5 = equal changes of isotactic or syndiotactic insertions (i.e. atactic)
- **rotation** – degrees to rotate monomer per insertion
- **md_spacing** – how many monomer insertion steps to perform between MD relaxation steps (1)
- **error_check** – True/False for if monomers should be checked for hardcore overlaps after insertion

Returns new polymer *System*

pysimm.apps.random_walk.**__check_tags__**(*m*, ***kwargs*)

pysimm.apps.random_walk.__check_tags__ private method to assert the polymerisation-related decorators assigned to the system 'm' that represents the next repetitive unit

pysimm.apps.random_walk.**check_tacticity**(*s*, *char_idxs*, *mon_len*)

pysimm.apps.random_walk.check_tacticity Method evaluates the local geometry of the polymer *System*. correct input includes :param char_idxs: characteristic indexes that define the structure of repetetive unit of the monomer. :type char_idxs: list of int :param It is supposed to have 4 elements which define index of: :type It is supposed to have 4 elements which define index of: 1) first atom in backbone; (2 :param backbone;: :type backbone;: 3) closest to backbone atom on the fisrt side chain; (4 :param side chain: :param mon_len: number of atoms in uncapped rep. unit :type mon_len: int

Note: currentely it is assumed that polymerisation does not change local indexing so indexes of corresponding characteristic atoms of the chain can be found by adding a number multiple of mon_len

Returns angles (in deg) between corresponding pairs of backbone vector (1-2) and normal to the plane produced by to side chains (2-3 x 2-4). Those vectors can be either on one half-space of (3-2-4) plane, so the angle will be >90 (deg) or on the opposite half-spaces of the plane, so the angle <90 (deg). orientations (list of boolean): sequence that tracks local geometry of a chain: records True if two consecutive rep.units form a meso dyad, and False if they form a racemo dyad

Return type angles (list of float)

pysimm.apps.zeopp

Module Contents

Functions

network(s, **kwargs)	pysimm.apps.zeopp.network

Attributes

ZEOpp_EXEC

`pysimm.apps.zeopp.ZEOpp_EXEC`

`pysimm.apps.zeopp.network`(*s*, ***kwargs*)

pysimm.apps.zeopp.network

Perform 1. Pore diameters; 2. Channel identification and dimensionality; 3. Surface area;

4. Accessible volume; 5. Pore size distribution calculation using zeo++ v2.2

with options to do 6. Probe-occupiable volume; 7. Stochastic ray tracing; 8. Blocking spheres;

9. Distance grids; 10. Structure analysis

Parameters

- `s` – pysimm System object or filename of file in CSSR | CUC | V1 | CIF format
- `atype_name` – True to use atom type as atom name (usually need radii and mass info), False to use atom element
- `radii` – file name that contain atom radii data (rad.rad)
- `mass` – file name that contain atom mass data (mass.mass)
- `probe_radius` – radius of a probe used in sampling of surface (1.2 A)
- `chan_radius` – radius of a probe used to determine accessibility of void space (1.2 A)
- `num_samples` – number of Monte Carlo samples per unit cell (50000)
- `simulation` (*option to include in the*) – set True to activate ha: default=True, for using high accuracy, res: default=True, for diameters of the largest included sphere, the largest free sphere and the largest included sphere along free sphere path chan: default=True, for channel systems characterized by dimensionality as well as Di, Df and Dif sa: default=True, for surface area accessible to a spherical probe, characterized by

  accessible surface area (ASA) and non-accessible surface area (NASA)

  vol: default=True, for accessible volume (AV) and non-accessible volume (NAV) volpo: default=False, for accessible proce-occupiable volume (POAV) and non-accessible probe-occupiable volume (PONAV) psd: default=True, for the "deriviative distribution" (change of AV w.r.t probe size) reported in the histogram file with 1000 bins of size of 0.1 Ang ray_atom: default=False block: default=False extra: user provided options, such as -gridG, -gridBOV, -strinfo, -oms, etc.

ZEOpp_EXEC: path to zeo++ executable (network)

Returns None

`pysimm.forcefield`

Submodules

`pysimm.forcefield.charmm`

Module Contents

Classes

`Charmm`	pysimm.forcefield.Charmm

Functions

`__parse_charmm__`()	Private method to read/convert CHARMM specific FF parameters from the form of GROMACS input format (.atp, .itp)
`__detect_rings__`(particle, orders)	Private method for analysing whether a given particle is a part of a ring structure

`class pysimm.forcefield.charmm.`**`Charmm`**(*db_file=None*)

Bases: *`pysimm.forcefield.forcefield.Forcefield`*

pysimm.forcefield.Charmm

Forcefield object with typing rules for CHARMM model. By default reads data file in forcefields subdirectory.

`ff_name`
charmm

`pair_style`
lj/charmm

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_ptypes

Charmm specific particle typing rules. Requires *`System`* object *`Particle`* objects have bonds defined. * **use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters **s** – *`System`*

Returns None

`assign_extra_ljtypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_extra_ljtypes

Addition to normal force field setup: filling up the non-diagonal interaction pair coefficients (coefficients for interaction of particles of different type).

Assumes that all *`ParticleType`* are defined for all particles in s

Parameters `s` – *System*

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_btypes

Gaff specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_atypes

Gaff specific boanglend typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_dtypes

CHARMM specific dihedral typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_itypes

Gaff specific improper typing rules. There are none.

Parameters `s` – *System*

Returns None

`assign_charges`(*self*, *s*, *charges='gasteiger'*)
pysimm.forcefield.Charmm.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- `s` – *System*
- `charges` – gasteiger

Returns None

`__parse_add_file__`(*self*, *file*)
Private method to read/convert CHARMM specific FF parameters from the native format (.prm) to add on top of currentely existing library of FF parameters. Will update this ForceField object with data from the file and will write the output 'charmm_mod.json' DB file

Parameters `file` – (string) full (absolute or relative) path to an .prm file

Returns none

`pysimm.forcefield.charmm.`**`__parse_charmm__`**`()`

Private method to read/convert CHARMM specific FF parameters from the form of GROMACS input format (.atp, .itp) to the PySIMM input format (.json).

Note: Because of the format specification, there are no sigma_{14} or epsilon_{14} parameters in the file as well as explicit non-diagonal LJ parameters (NBFIXes). They are read from a different file types (see charmm.__parse_add_file__())

Returns None

`pysimm.forcefield.charmm.`**`__detect_rings__`**`(`*particle*, *orders*`)`

Private method for analysing whether a given particle is a part of a ring structure

Parameters

- **`particle`** – *`Particle`* reference
- **`orders`** – list of integers defining size of the rings which should be checked

Returns list of integers subset of orders which defines the sizes of the rings that contain particle; returns 0 if no cyclic structures of size orders are detected

`pysimm.forcefield.dreiding`

Module Contents

Classes

`Dreiding`	pysimm.forcefield.Dreiding

class `pysimm.forcefield.dreiding.`**`Dreiding`**`(`*db_file=None*`)`

Bases: *`pysimm.forcefield.forcefield.Forcefield`*

pysimm.forcefield.Dreiding

Forcefield object with typing rules for Dreiding model. By default reads data file in forcefields subdirectory.

`ff_name`
dreiding

`pair_style`
lj

`ff_class`
1

`assign_ptypes``(`*self*, *s*`)`

pysimm.forcefield.Dreiding.assign_ptypes

Dreiding specific particle typing rules. Requires *`System`* object *`Particle`* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

Parameters **`s`** – *`System`*

Returns None

`assign_btypes``(`*self*, *s*`)`

pysimm.forcefield.Dreiding.assign_btypes

Dreiding specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – *System*

Returns None

assign_atypes(*self*, *s*)

pysimm.forcefield.Dreiding.assign_atypes

Dreiding specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – *System*

Returns None

assign_dtypes(*self*, *s*)

pysimm.forcefield.Dreiding.assign_dtypes

Dreiding specific dihedral typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – *System*

Returns None

assign_itypes(*self*, *s*)

pysimm.forcefield.Dreiding.assign_itypes

Dreiding specific improper typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – *System*

Returns None

assign_charges(*self*, *s*, *charges='gasteiger'*)

pysimm.forcefield.Dreiding.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – *System*
- **charges** – gasteiger

Returns None

pysimm.forcefield.forcefield

Module Contents

Classes

Forcefield	pysimm.forcefield.Forcefield

Attributes

element_names_by_mass	

pysimm.forcefield.forcefield.**element_names_by_mass**

class pysimm.forcefield.forcefield.**Forcefield**(*file_=None*, *format=None*)

Bases: `object`

pysimm.forcefield.Forcefield

Base Forcefield class definition. Initialize with force field xml file.

ff_class
force field class (1 or 2)

ff_name
force field name

particle_types
ItemContainer for particle_types

bond_types
ItemContainer for bond_types

angle_types
ItemContainer for angle_types

dihedral_types
ItemContainer for dihedral_types

improper_types
ItemContainer for improper_types

from_xml(*self*, *file_*)

from_json(*self*, *json_file*)

write_json(*self*, *out*)

write_xml(*self*, *out*)

pysimm.forcefield.Forcefield.write

Write Forcefield object to xml format.

Parameters **out** – file name to write

Returns None

pysimm.forcefield.gaff

Module Contents

Classes

Gaff	pysimm.forcefield.Gaff

class `pysimm.forcefield.gaff.`**`Gaff`**(*db_file=None*)

Bases: `pysimm.forcefield.forcefield.Forcefield`

pysimm.forcefield.Gaff

Forcefield object with typing rules for Gaff model. By default reads data file in forcefields subdirectory.

`ff_name`
gaff

`pair_style`
lj

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_ptypes

Gaff specific particle typing rules. Requires `System` object `Particle` objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters `s` – `System`

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_btypes

Gaff specific bond typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_atypes

Gaff specific boanglend typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_dtypes

Gaff specific dihedral typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_itypes

Gaff specific improper typing rules. There are none.

Parameters `s` – `System`

Returns None

assign_charges(*self*, *s*, *charges='gasteiger'*)
pysimm.forcefield.Gaff.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – *System*
- **charges** – gasteiger

Returns None

pysimm.forcefield.gaff2

Module Contents

Classes

Gaff2	pysimm.forcefield.Gaff2

class pysimm.forcefield.gaff2.**Gaff2**(*db_file=None*)
Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Gaff2

Forcefield object with typing rules for Gaff2 model. By default reads data file in forcefields subdirectory.

ff_name
gaff2

pair_style
lj

bond_style
harmonic

angle_style
harmonic

dihedral_style
fourier

improper_style
cvff

ff_class
1

assign_ptypes(*self*, *s*)
pysimm.forcefield.Gaff2.assign_ptypes

Gaff2 specific particle typing rules. Requires *System* object *Particle* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters **s** – *System*

Returns None

assign_btypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_btypes

Gaff2 specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_atypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_atypes

Gaff2 specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_dtypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_dtypes

Gaff2 specific dihedral typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_itypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_itypes

Gaff2 specific improper typing rules.

Parameters **s** – *System*

Returns None

assign_charges(*self*, *s*, *charges='gasteiger'*)

pysimm.forcefield.Gaff.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – *System*
- **charges** – gasteiger

Returns None

pysimm.forcefield.gasteiger

Module Contents

Functions

set_charges(s, maxiter=100, tol=1e-06)	

Attributes

element_names_by_mass
gasteiger_parameters

pysimm.forcefield.gasteiger.**element_names_by_mass**

pysimm.forcefield.gasteiger.**gasteiger_parameters**

pysimm.forcefield.gasteiger.**set_charges**(*s*, *maxiter=100*, *tol=1e-06*)

pysimm.forcefield.pcff

Module Contents

Classes

Pcff	pysimm.forcefield.Pcff

class pysimm.forcefield.pcff.**Pcff**(*db_file=None*)

Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Pcff

Forcefield object with typing rules for Pcff model. By default reads data file in forcefields subdirectory.

ff_name
pcff

pair_style
class2

ff_class
2

nb_mixing
sixth

assign_ptypes(*self*, *s*)
pysimm.forcefield.Pcff.assign_ptypes

Pcff specific particle typing rules. Requires *System* object *Particle* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

Parameters **s** – *System*

Returns None

assign_btypes(*self*, *s*)
pysimm.forcefield.Pcff.assign_btypes

Pcff specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_atypes

Pcff specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_dtypes

Pcff specific dihedral typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_itypes

Pcff specific improper typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

`assign_charges`(*self*, *s*, *charges='default'*)
pysimm.forcefield.Pcff.assign_charges

Default Pcff charge assignment. Gasteiger is also an option.

Parameters

- **s** – *System*
- **charges** – default

Returns None

`pysimm.forcefield.tip3p`

Module Contents

Classes

Tip3p	pysimm.forcefield.Tip3p

class pysimm.forcefield.tip3p.**Tip3p**(*db_file=None*)
Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Tip3p

Forcefield object with typing rules for Tip3p model. By default reads data file in forcefields subdirectory.

`ff_name`
tip3p

`pair_style`
lj

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_ptypes

Tip3p specific particle typing rules. Requires `System` object `Particle` objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

Parameters `s` – `System`

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_btypes

Tip3p specific bond typing rules. Requires `System` object `Particle` objects have type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_atypes

Tip3p specific angle typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_dtypes

Tip3p specific dihedral typing rules. There are none.

Parameters `s` – `System`

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_itypes

Tip3p specific improper typing rules. There are none.

Parameters `s` – `System`

Returns None

`assign_charges`(*self*, *s*, *charges='default'*)
pysimm.forcefield.Tip3p.assign_charges

Tip3p specific charge assignment. There are none.

Parameters

- `s` – `System`
- `charges` – default

Returns None

Package Contents

Classes

Forcefield	pysimm.forcefield.Forcefield
Dreiding	pysimm.forcefield.Dreiding
Charmm	pysimm.forcefield.Charmm
Gaff	pysimm.forcefield.Gaff
Gaff2	pysimm.forcefield.Gaff2
Pcff	pysimm.forcefield.Pcff
Tip3p	pysimm.forcefield.Tip3p

Functions

__parse_charmm__()	Private method to read/convert CHARMM specific FF parameters from the form of GROMACS input format (.atp, .itp)

class pysimm.forcefield.**Forcefield**(*file_=None*, *format=None*)

Bases: object

pysimm.forcefield.Forcefield

Base Forcefield class definition. Initialize with force field xml file.

ff_class
force field class (1 or 2)

ff_name
force field name

particle_types
ItemContainer for particle_types

bond_types
ItemContainer for bond_types

angle_types
ItemContainer for angle_types

dihedral_types
ItemContainer for dihedral_types

improper_types
ItemContainer for improper_types

from_xml(*self*, *file_*)

from_json(*self*, *json_file*)

write_json(*self*, *out*)

write_xml(*self*, *out*)
pysimm.forcefield.Forcefield.write

Write Forcefield object to xml format.

Parameters `out` – file name to write

Returns None

`class pysimm.forcefield.Dreiding`(*db_file=None*)

Bases: *`pysimm.forcefield.forcefield.Forcefield`*

pysimm.forcefield.Dreiding

Forcefield object with typing rules for Dreiding model. By default reads data file in forcefields subdirectory.

`ff_name`
dreiding

`pair_style`
lj

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Dreiding.assign_ptypes

Dreiding specific particle typing rules. Requires *`System`* object *`Particle`* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

Parameters `s` – *`System`*

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Dreiding.assign_btypes

Dreiding specific bond typing rules. Requires *`System`* object *`Particle`* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *`System`*

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Dreiding.assign_atypes

Dreiding specific angle typing rules. Requires *`System`* object *`Particle`* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *`System`*

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Dreiding.assign_dtypes

Dreiding specific dihedral typing rules. Requires *`System`* object *`Particle`* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *`System`*

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Dreiding.assign_itypes

Dreiding specific improper typing rules. Requires *`System`* object *`Particle`* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *`System`*

Returns None

`assign_charges`(*self*, *s*, *charges='gasteiger'*)
pysimm.forcefield.Dreiding.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – *System*
- **charges** – gasteiger

Returns None

`class pysimm.forcefield.Charmm`(*db_file=None*)
Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Charmm

Forcefield object with typing rules for CHARMM model. By default reads data file in forcefields subdirectory.

`ff_name`
charmm

`pair_style`
lj/charmm

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_ptypes

Charmm specific particle typing rules. Requires *System* object *Particle* objects have bonds defined. * **use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters **s** – *System*

Returns None

`assign_extra_ljtypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_extra_ljtypes

Addition to normal force field setup: filling up the non-diagonal interaction pair coefficients (coefficients for interaction of particles of different type).

Assumes that all *ParticleType* are defined for all particles in s

Parameters **s** – *System*

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Charmm.assign_btypes

Gaff specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_atypes(*self*, *s*)

pysimm.forcefield.Charmm.assign_atypes

Gaff specific boanglend typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – `System`

Returns None

assign_dtypes(*self*, *s*)

pysimm.forcefield.Charmm.assign_dtypes

CHARMM specific dihedral typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – `System`

Returns None

assign_itypes(*self*, *s*)

pysimm.forcefield.Charmm.assign_itypes

Gaff specific improper typing rules. There are none.

Parameters **s** – `System`

Returns None

assign_charges(*self*, *s*, *charges='gasteiger'*)

pysimm.forcefield.Charmm.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – `System`
- **charges** – gasteiger

Returns None

__parse_add_file__(*self*, *file*)

Private method to read/convert CHARMM specific FF parameters from the native format (.prm) to add on top of currentely existing library of FF parameters. Will update this ForceField object with data from the file and will write the output 'charmm_mod.json' DB file

Parameters **file** – (string) full (absolute or relative) path to an .prm file

Returns none

pysimm.forcefield.**__parse_charmm__**()

Private method to read/convert CHARMM specific FF parameters from the form of GROMACS input format (.atp, .itp) to the PySIMM input format (.json).

Note: Because of the format specification, there are no sigma_{14} or epsilon_{14} parameters in the file as well as explicit non-diagonal LJ parameters (NBFIXes). They are read from a different file types (see charmm.__parse_add_file__())

Returns None

class pysimm.forcefield.**Gaff**(*db_file=None*)

Bases: `pysimm.forcefield.forcefield.Forcefield`

pysimm.forcefield.Gaff

Forcefield object with typing rules for Gaff model. By default reads data file in forcefields subdirectory.

`ff_name`
gaff

`pair_style`
lj

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_ptypes

Gaff specific particle typing rules. Requires `System` object `Particle` objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters `s` – `System`

Returns None

`assign_btypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_btypes

Gaff specific bond typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_atypes

Gaff specific boanglend typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_dtypes

Gaff specific dihedral typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – `System`

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Gaff.assign_itypes

Gaff specific improper typing rules. There are none.

Parameters `s` – `System`

Returns None

`assign_charges`(*self*, *s*, *charges='gasteiger'*)
pysimm.forcefield.Gaff.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – *System*
- **charges** – gasteiger

Returns None

class pysimm.forcefield.**Gaff2**(*db_file=None*)

Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Gaff2

Forcefield object with typing rules for Gaff2 model. By default reads data file in forcefields subdirectory.

ff_name
gaff2

pair_style
lj

bond_style
harmonic

angle_style
harmonic

dihedral_style
fourier

improper_style
cvff

ff_class
1

assign_ptypes(*self*, *s*)
pysimm.forcefield.Gaff2.assign_ptypes

Gaff2 specific particle typing rules. Requires *System* object *Particle* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

*** Not entirely inclusive - some atom types not used ***

Parameters **s** – *System*

Returns None

assign_btypes(*self*, *s*)
pysimm.forcefield.Gaff2.assign_btypes

Gaff2 specific bond typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_atypes(*self*, *s*)
pysimm.forcefield.Gaff2.assign_atypes

Gaff2 specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_dtypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_dtypes

Gaff2 specific dihedral typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters **s** – `System`

Returns None

assign_itypes(*self*, *s*)

pysimm.forcefield.Gaff2.assign_itypes

Gaff2 specific improper typing rules.

Parameters **s** – `System`

Returns None

assign_charges(*self*, *s*, *charges='gasteiger'*)

pysimm.forcefield.Gaff.assign_charges

Charge assignment. Gasteiger is default for now.

Parameters

- **s** – `System`
- **charges** – gasteiger

Returns None

class pysimm.forcefield.**Pcff**(*db_file=None*)

Bases: `pysimm.forcefield.forcefield.Forcefield`

pysimm.forcefield.Pcff

Forcefield object with typing rules for Pcff model. By default reads data file in forcefields subdirectory.

ff_name

pcff

pair_style

class2

ff_class

2

nb_mixing

sixth

assign_ptypes(*self*, *s*)

pysimm.forcefield.Pcff.assign_ptypes

Pcff specific particle typing rules. Requires `System` object `Particle` objects have bonds defined. * **use System.add_particle_bonding() to ensure this** *

Parameters **s** – `System`

Returns None

assign_btypes(*self*, *s*)

pysimm.forcefield.Pcff.assign_btypes

Pcff specific bond typing rules. Requires `System` object `Particle` objects have bonds, type and type.name defined. * **use after assign_ptypes** *

Parameters `s` – *System*

Returns None

`assign_atypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_atypes

Pcff specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_dtypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_dtypes

Pcff specific dihedral typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_itypes`(*self*, *s*)
pysimm.forcefield.Pcff.assign_itypes

Pcff specific improper typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters `s` – *System*

Returns None

`assign_charges`(*self*, *s*, *charges='default'*)
pysimm.forcefield.Pcff.assign_charges

Default Pcff charge assignment. Gasteiger is also an option.

Parameters

- `s` – *System*
- `charges` – default

Returns None

class pysimm.forcefield.**Tip3p**(*db_file=None*)
Bases: *pysimm.forcefield.forcefield.Forcefield*

pysimm.forcefield.Tip3p

Forcefield object with typing rules for Tip3p model. By default reads data file in forcefields subdirectory.

`ff_name`
tip3p

`pair_style`
lj

`ff_class`
1

`assign_ptypes`(*self*, *s*)
pysimm.forcefield.Tip3p.assign_ptypes

Tip3p specific particle typing rules. Requires *System* object *Particle* objects have bonds defined. *** use System.add_particle_bonding() to ensure this ***

Parameters **s** – *System*

Returns None

assign_btypes(*self*, *s*)
pysimm.forcefield.Tip3p.assign_btypes

Tip3p specific bond typing rules. Requires *System* object *Particle* objects have type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_atypes(*self*, *s*)
pysimm.forcefield.Tip3p.assign_atypes

Tip3p specific angle typing rules. Requires *System* object *Particle* objects have bonds, type and type.name defined. *** use after assign_ptypes ***

Parameters **s** – *System*

Returns None

assign_dtypes(*self*, *s*)
pysimm.forcefield.Tip3p.assign_dtypes

Tip3p specific dihedral typing rules. There are none.

Parameters **s** – *System*

Returns None

assign_itypes(*self*, *s*)
pysimm.forcefield.Tip3p.assign_itypes

Tip3p specific improper typing rules. There are none.

Parameters **s** – *System*

Returns None

assign_charges(*self*, *s*, *charges='default'*)
pysimm.forcefield.Tip3p.assign_charges

Tip3p specific charge assignment. There are none.

Parameters

- **s** – *System*
- **charges** – default

Returns None

pysimm.models

Subpackages

pysimm.models.monomers

Subpackages

pysimm.models.monomers.dreiding

Submodules

pysimm.models.monomers.dreiding.NbTMS_H2_tacticity

Module Contents

Functions

monomer(**kwargs)	
polymer_chain(length)	

pysimm.models.monomers.dreiding.NbTMS_H2_tacticity.**monomer**(***kwargs*)

pysimm.models.monomers.dreiding.NbTMS_H2_tacticity.**polymer_chain**(*length*)

pysimm.models.monomers.dreiding.pe

Module Contents

Functions

monomer()	
polymer_chain(length)	
polymer_system(chains=10, mn=1000, pdi=1, density=0.3)	

pysimm.models.monomers.dreiding.pe.**monomer**()

pysimm.models.monomers.dreiding.pe.**polymer_chain**(*length*)

pysimm.models.monomers.dreiding.pe.**polymer_system**(*chains=10*, *mn=1000*, *pdi=1*, *density=0.3*)

pysimm.models.monomers.dreiding.pmma

Module Contents

Functions

monomer()	
polymer_chain(length)	

pysimm.models.monomers.dreiding.pmma.**monomer**()

pysimm.models.monomers.dreiding.pmma.**polymer_chain**(*length*)

pysimm.models.monomers.dreiding.ps

Module Contents

Functions

monomer(is_capped=False)	
polymer_chain(length)	

pysimm.models.monomers.dreiding.ps.**monomer**(*is_capped=False*)

pysimm.models.monomers.dreiding.ps.**polymer_chain**(*length*)

pysimm.models.monomers.ff_typers

Submodules

pysimm.models.monomers.ff_typers.pe

Module Contents

Functions

monomer(ff, is_capped=False)	
polymer_chain(length, ff)	

pysimm.models.monomers.ff_typers.pe.**monomer**(*ff*, *is_capped=False*) pysimm.models.monomers.ff_types.pe.**polymer_chain**(*length*, *ff*)

pysimm.models.monomers.ff_typers.pmma

Module Contents

Functions

monomer(ff, is_capped=False)	
polymer_chain(length, ff)	

pysimm.models.monomers.ff_typers.pmma.**monomer**(*ff*, *is_capped=False*)

pysimm.models.monomers.ff_typers.pmma.**polymer_chain**(*length*, *ff*)

pysimm.models.monomers.ff_typers.ps

Module Contents

Functions

monomer(ff, is_capped=False)	
polymer_chain(length, ff)	

pysimm.models.monomers.ff_typers.ps.**monomer**(*ff*, *is_capped=False*)

pysimm.models.monomers.ff_typers.ps.**polymer_chain**(*length*, *ff*)

pysimm.models.monomers.gaff

Submodules

pysimm.models.monomers.gaff.pe

Module Contents

Functions

monomer()	

continues on next page

Table 31 – continued from previous page

polymer_chain(length)	

pysimm.models.monomers.gaff.pe.**monomer**()

pysimm.models.monomers.gaff.pe.**polymer_chain**(*length*)

pysimm.models.monomers.gaff.pmma

Module Contents

Functions

monomer()	
polymer_chain(length)	

pysimm.models.monomers.gaff.pmma.**monomer**()

pysimm.models.monomers.gaff.pmma.**polymer_chain**(*length*)

pysimm.models.monomers.gaff.ps

Module Contents

Functions

monomer()	
polymer_chain(length)	

pysimm.models.monomers.gaff.ps.**monomer**()

pysimm.models.monomers.gaff.ps.**polymer_chain**(*length*)

pysimm.models.monomers.gaff2

Submodules

pysimm.models.monomers.gaff2.pe

Module Contents

Functions

monomer()	
polymer_chain(length)	

`pysimm.models.monomers.gaff2.pe.`**`monomer`**`()`

`pysimm.models.monomers.gaff2.pe.`**`polymer_chain`**`(`*length*`)`

pysimm.models.monomers.gaff2.pmma

Module Contents

Functions

monomer()	
polymer_chain(length)	

`pysimm.models.monomers.gaff2.pmma.`**`monomer`**`()`

`pysimm.models.monomers.gaff2.pmma.`**`polymer_chain`**`(`*length*`)`

pysimm.models.monomers.gaff2.ps

Module Contents

Functions

monomer()	
polymer_chain(length)	

`pysimm.models.monomers.gaff2.ps.`**`monomer`**`()`

`pysimm.models.monomers.gaff2.ps.`**`polymer_chain`**`(`*length*`)` `pysimm.models.monomers.topologies`

1.1.2 Submodules

`pysimm.amber`

Module Contents

Functions

cleanup_antechamber()	pysimm.amber.cleanup_antechamber
calc_charges(s, charge_method='bcc', cleanup=True)	pysimm.amber.calc_charges
get_forcefield_types(s, types='gaff', f=None)	pysimm.amber.get_forcefield_types

Attributes

ANTECHAMBER_EXEC

pysimm.amber.**ANTECHAMBER_EXEC**

pysimm.amber.**cleanup_antechamber**()

pysimm.amber.cleanup_antechamber

Removes temporary files created by antechamber and pysimm.

Parameters **None** –

Returns None

pysimm.amber.**calc_charges**(*s*, *charge_method='bcc'*, *cleanup=True*)

pysimm.amber.calc_charges

Calculates charges using antechamber. Defaults to am1-bcc charges.

Parameters

- **s** – System for which to calculate charges. System object is updated in place
- **charge_method** – name of charge derivation method to use (default: bcc)
- **cleanup** – removes temporary files created by antechamber (default: True)

Returns None

pysimm.amber.**get_forcefield_types**(*s*, *types='gaff'*, *f=None*)

pysimm.amber.get_forcefield_types

Uses antechamber to determine atom types. Defaults to GAFF atom types. Retrieves *ParticleType* objects from force field is provided

Parameters

- **s** – *System* for which to type
- **types** – name of atom types to use (default: gaff)

- **f** – forcefield object to retrieve *ParticleType* objects from if not present in s (default: None)

Returns None

pysimm.calc

Module Contents

Functions

intersection(line1, line2)	pysimm.calc.intersection
find_rotation(a, b)	pysimm.calc.find_rotation
rotate_vector(x, y, z, theta_x=None, theta_y=None, theta_z=None)	pysimm.calc.rotate_vector
distance(p1, p2)	pysimm.calc.distance
angle(p1, p2, p3, radians=False)	pysimm.calc.angle
dihedral(p1, p2, p3, p4, radians=False)	
chiral_angle(a, b, c, d)	pysimm.calc.chiral_angle
tacticity(s, a_tag=None, b_tag=None, c_tag=None, d_tag=None, offset=None, return_angles=True, unwrap=True, rewrap=True, skip_first=False)	pysimm.calc.tacticity
frac_free_volume(v_sp, v_void)	pysimm.calc.frac_free_volume
pbc_distance(s, p1, p2)	pysimm.calc.pbc_distance
LJ_12_6(pt, d)	
LJ_9_6(pt, d)	
buckingham(pt, d)	
harmonic_bond(bt, d)	
class2_bond(bt, d)	
harmonic_angle(at, d)	
class2_angle(at, d)	
harmonic_dihedral(dt, d)	
class2_dihedral(dt, d)	
opls_dihedral(dt, d)	
fourier_dihedral(dt, d)	
harmonic_improper(it, d)	
cvff_improper(it, d)	

continues on next page

Table 39 – continued from previous page

umbrella_improper(it, d)	

Attributes

np	

`pysimm.calc.np`

`pysimm.calc.intersection`(*line1*, *line2*)

pysimm.calc.intersection

Finds intersection between two 2D lines given by two sets of points

Parameters

- **line1** – [[x1,y1], [x2,y2]] for line 1
- **line2** – [[x1,y1], [x2,y2]] for line 2

Returns x,y intersection point

`pysimm.calc.find_rotation`(*a*, *b*)

pysimm.calc.find_rotation

Finds rotation vector required to align vector a and vector b

Parameters

- **a** – 3D vector [x,y,z]
- **b** – 3D vector [x,y,z]

Returns rotation matrix

`pysimm.calc.rotate_vector`(*x*, *y*, *z*, *theta_x=None*, *theta_y=None*, *theta_z=None*)

pysimm.calc.rotate_vector

Rotates 3d vector around x-axis, y-axis and z-axis given by user defined angles

Parameters

- **x** – x vector component
- **y** – y vector component
- **z** – z vector component
- **theta_x** – angle to rotate vector around x axis
- **theta_y** – angle to rotate vector around y axis
- **theta_z** – angle to rotate vector around z axis

Returns new vector [x,y,z]

`pysimm.calc.distance`(*p1*, *p2*)

pysimm.calc.distance

Finds distance between two *Particle* objects. Simply calculates length of vector between particle coordinates and does not consider periodic boundary conditions.

Parameters

- **p1** – *Particle*
- **p2** – *Particle*

Returns distance between particles

`pysimm.calc.`**`angle`**(*p1*, *p2*, *p3*, *radians=False*)

pysimm.calc.angle

Finds angle between three *Particle* objects. Does not consider periodic boundary conditions.

Parameters

- **p1** – pysimm.system.Particle
- **p2** – pysimm.system.Particle
- **p3** – pysimm.system.Particle
- **radians** – returns value in radians if True (False)

Returns angle between particles

`pysimm.calc.`**`dihedral`**(*p1*, *p2*, *p3*, *p4*, *radians=False*)

`pysimm.calc.`**`chiral_angle`**(*a*, *b*, *c*, *d*)

pysimm.calc.chiral_angle

Finds chiral angle between four *Particle* objects. Chiral angle is defined as the angle between the vector resulting from vec(a->c) X vec(a->d) and vec(a->b). Used to help define tacticity where backbone follow b'–a–b and c and d are side groups.

b'–a–b / c d

Parameters

- **a** – pysimm.system.Particle
- **b** – pysimm.system.Particle
- **c** – pysimm.system.Particle
- **d** – pysimm.system.Particle

Returns chiral angle

`pysimm.calc.`**`tacticity`**(*s*, *a_tag=None*, *b_tag=None*, *c_tag=None*, *d_tag=None*, *offset=None*, *return_angles=True*, *unwrap=True*, *rewrap=True*, *skip_first=False*)

pysimm.calc.tacticity

Determines tacticity for polymer chain. Iterates through groups of four patricles given by X_tags, using offset. This assumes equivalent atoms in each group of four are perfectly offset.

Parameters

- **s** – *System*
- **a_tag** – tag of first a particle
- **b_tag** – tag of first b particle
- **c_tag** – tag of first c particle
- **d_tag** – tag of first d particle
- **offset** – offset of particle tags (monomer repeat atomic count)

- **return_angles** – if True return chiral angles of all monomers
- **unwrap** – True to perform unwrap before calculation (REQUIRED before calculation, but not required in this
- **function)** –
- **rewrap** – True to rewrap system after calculation
- **skip_first** – True to skip first monomer (sometime chirality is poorly defined for thsi monomer)

Returns tacticity or tacticity, [chiral_angles]

pysimm.calc.**frac_free_volume**(*v_sp*, *v_void*)

pysimm.calc.frac_free_volume

Determines fractional free volume for a poroous system.

Parameters

- **v_sp** – specific volume
- **v_void** – void volume

Returns fractional free volume

pysimm.calc.**pbc_distance**(*s*, *p1*, *p2*)

pysimm.calc.pbc_distance

Calculates distance between particles using PBC

Parameters

- **s** – *System*
- **p1** – *Particle*
- **p2** – *Particle*

Returns distance between particles

pysimm.calc.**LJ_12_6**(*pt*, *d*)

pysimm.calc.**LJ_9_6**(*pt*, *d*)

pysimm.calc.**buckingham**(*pt*, *d*)

pysimm.calc.**harmonic_bond**(*bt*, *d*)

pysimm.calc.**class2_bond**(*bt*, *d*)

pysimm.calc.**harmonic_angle**(*at*, *d*)

pysimm.calc.**class2_angle**(*at*, *d*)

pysimm.calc.**harmonic_dihedral**(*dt*, *d*)

pysimm.calc.**class2_dihedral**(*dt*, *d*)

pysimm.calc.**opls_dihedral**(*dt*, *d*)

pysimm.calc.**fourier_dihedral**(*dt*, *d*)

pysimm.calc.**harmonic_improper**(*it*, *d*)

pysimm.calc.**cvff_improper**(*it*, *d*)

pysimm.calc.**umbrella_improper**(*it*, *d*)

`pysimm.cassandra`

Module Contents

Classes

MCSimulation	pysimm.cassandra.MCSimulation
GCMC	pysimm.cassandra.GCMC
NVT	pysimm.cassandra.NVT
NPT	pysimm.cassandra.NPT
InpSpec	pysimm.cassandra.InpSpec
InpProbSpec	pysimm.cassandra.InpSpec
McSystem	pysimm.cassandra.McSystem
Cassandra	pysimm.cassandra.Cassandra
McfWriter	pysimm.cassandra.McfWriter

Functions

check_cs_exec()	pysimm.cassandra.check_cs_exec
make_iterable(obj)	pysimm.cassandra.make_iterable

Attributes

DATA_PATH
KCALMOL_2_K
CASSANDRA_EXEC
DEFAULT_PARAMS

`pysimm.cassandra.`**`DATA_PATH`**

`pysimm.cassandra.`**`KCALMOL_2_K = 503.22271716452`**

`pysimm.cassandra.`**`CASSANDRA_EXEC`**

`pysimm.cassandra.`**`DEFAULT_PARAMS`**

`class` `pysimm.cassandra.`**`MCSimulation`**(*mc_sst=None*, *init_sst=None*, ***kwargs*)

Bases: `object`

pysimm.cassandra.MCSimulation

Object containing the settings and the logic necessary to partially set-up an abstract Monte Carlo simulation to be submitted to the CASSANDRA software. The object also will include the simulation results once the simulations are finished.

`mc_sst`

describes all molecules to be inserted by CASSANDRA

Type *McSystem*

`init_sst`

describes the optional initial fixed molecular configuration for MC simulations (default: empty cubic box with 1 nm side length). If the particles in the system are not attributed with the flag *is_fixed* all of them are considered to be fixed, and will be marked with this flag, otherwise all particles with is_fixed=False will be removed.

Type *System*

Keyword Arguments

- **out_folder** (*str*) – the relative path of the simulation results (all .dat, .mcf, as well as .chk, … files will go there). If the folder does not exist it will be created with 0755 permissions.
- **props_file** (*str*) – the name of the .inp file.

Note: Other keyword arguments that are accepted are the GCMC simulation settings. The keywords of the settings are the same as they are described in CASSANDRA specification but without # symbol.

For example: the keyword argument *Run_Name='my_simulation'* will set *#Run_Name* setting in CASSANDRA input file to *my_simulation* value

---

Parameters

- **props** (*dictionary*) – include all simulation settings to be written to the CASSANDRA .inp file
- **input** (*str*) – text stream that will be written to the CASSANDRA .inp file
- **tot_sst** (*System*) – object containing the results of CASSANDRA simulations

`write`(*self*)

pysimm.cassandra.MCSimulation.write

Iterates through the `props` dictionary creating the text for correct CASSANDRA input

`group_by_id`(*self*, *group_key='matrix'*)

pysimm.cassandra.MCSimulation.group_by_id

Method groups the atoms of the system `tot_sst` by a certain property. Will iterate through all atoms in the system and return indexes of only those atoms that match the property. Currently supports 3 properties defined by the input keyword argument argument.

Keyword Arguments **group_key** (*str*) – text constant defines the property to match. Possible keywords are:

(1) *matrix* – (default) indexes of the atoms in `fxd_sst`

(2) *rigid* – indexes of all atoms that have rigid atomic bonds. It is assumed here that rigid and nonrigid atoms can interact only through intermolecular forces

(3) *nonrigid* – opposite of previous, indexes of all atoms that have nonrigid atomic bonds

Returns string in format *a1:b1 a2:b2 …* where all indexes inside *[ak, bk]* belongs to the selected group and array of the form *[[a1, b1], [a2, b2], …]*

Return type str

upd_simulation(*self*)
pysimm.cassandra.MCSimulation.upd_simulation

Updates the `tot_sst` field using the *MCSimulation.props['Run_Name'].chk* file. Will try to parse the checkpoint file and read the coordinates of the molecules inserted by CASSANDRA. If neither of the molecules from the *`mc_sst`* can be fit to the text that was read the method will raise an exception. The fitting method: *`make_system`* assumes that different molecules inserted by CASSANDRA have the same order of the atoms.

__check_params__(*self*)
pysimm.cassandra.MCSimulation.__check_params__

Private method designed for update the fields of the simulation object to make them conformed with each other

__write_chk__(*self*, *out_file*)
pysimm.cassandra.MCSimulation.__write_chk__

Creates the CASSANDRA checkpoint file basing on the information from the *~MCSimulation.tot_sst* field

get_prp(*self*)

class `pysimm.cassandra.`**GCMC**(*mc_sst=None*, *init_sst=None*, ***kwargs*)
Bases: *`MCSimulation`*

pysimm.cassandra.GCMC Initiates the specific type of Monte Carlo simulations for CASSANDRA: simulations using Grand-Canonical ensemble of particles (constant volume-temperature-chemical potential, muVT). See *`MCSimulation`* for the detailed description of the properties.

class `pysimm.cassandra.`**NVT**(*mc_sst=None*, *init_sst=None*, ***kwargs*)
Bases: *`MCSimulation`*

pysimm.cassandra.NVT Initiates the specific type of Monte Carlo simulations for CASSANDRA: simulations using Canonical ensemble of particles (constant volume-temperature-number of particles, NVT). See *`MCSimulation`* for the detailed description of the properties.

class `pysimm.cassandra.`**NPT**(*mc_sst=None*, *init_sst=None*, ***kwargs*)
Bases: *`MCSimulation`*

pysimm.cassandra.NPT Initiates the specific type of Monte Carlo simulations for CASSANDRA: simulations using Isobaric-Isothermal ensemble of particles (NPT). See *`MCSimulation`* for the detailed description of the properties.

class `pysimm.cassandra.`**InpSpec**(*key*, *value*, *default*, ***kwargs*)
Bases: `object`

pysimm.cassandra.InpSpec

Represents the most common object used for carrying one logical unit of the CASSANDRA simulation options

Parameters

- **key** (*str*) – the keyword of the simulation option (literally the string that goes after the # sign in CASSANDRA .inp file)
- **value** (*object*) – numerical or text values of the particular simulation option structured in a certain way. Here goes only the values that are wished to be changed (it might be just one field of a big dictionary)
- **default** (*object*) – the most complete default description of the simulation option

Keyword Arguments

- **write_headers** (*boolean*) – if the `value` is dictionary defines whether the dictionary keys should be written to the output
- **new_line** (*boolean*) – if the `value` is iterable defines whether each new element will be written to the new line

to_string(*self*)

pysimm.cassandra.InpSpec.to_string

Creates the proper text representation of the property stored in the `value` field

Returns formatted text string

Return type str

class pysimm.cassandra.**InpProbSpec**(*key*, *value*, *default*, ***kwargs*)

Bases: *InpSpec*

pysimm.cassandra.InpSpec

Extension of the *InpSpec* class that takes into account special representation of the movement probabilities in the CASSANDRA input file.

to_string(*self*)

pysimm.cassandra.InpSpec.to_string

Creates the proper text representation of the property stored in the `value` field

Returns formatted text string

Return type str

class pysimm.cassandra.**McSystem**(*sst*, ***kwargs*)

Bases: `object`

pysimm.cassandra.McSystem

Wrapper around the list of *System* objects. Each element in the list represents single molecule of a different specie that will be used during MC simulations. Additionally, the object is responsible for creating .dat and .mcf files needed for the simulation and reading back the CASSANDRA simulation results.

sst

items representing single molecules of different species to be inserted by CASSANDRA. If the sst is a list (not a single value) it is assumed that all of the following properties are synchronized with it by indexes.

Type list of *System*

chem_pot

chemical potential for each specie [Joule/mol]

Type list of int

Keyword Arguments

- **max_ins** (*list of int*) – defines the highest possible number of molecules of corresponding specie. Basing on these values CASSANDRA allocates memory for simulations. (default: 5000).
- **is_rigid** (*list of boolean*) – defines whether the atoms in the particular molecule should be marked as rigid or not. **Important!** In current implementation the module doesn't support flexible molecule angles, so the *is_rigid=False* is designed to be used exclusively for **single bead** molecules.

Parameters

- **made_ins** (*list of int*) – number of particles of each specie inserted by CASSANDRA.
- **mcf_file** (*list of str*) – defines full relative names of molecule configuration files **(.mcf)** required by CASSANDRA. Files will be created automatically.
- **frag_file** (*list of str*) – defines full relative names of possible relative configuration files **(.dat)** required by CASSANDRA. Files will be created automatically.

update_props(*self*, *props*)
pysimm.cassandra.McSystem.update_props

For each specie in the system creates the .mcf file required for CASSANDRA simulation.

Parameters **props** (*dictionary*) – contains the .mcf file names and maximally allowed number of molecules insertions. The dictionary is to be assigned to 'Molecule_Files' property of the MC simulation

Returns updated input dictionary

Return type props

update_frag_record(*self*, *frag_record*)
pysimm.cassandra.McSystem.update_frag_record

For each specie in the system creates the single configuration .dat file required for CASSANDRA simulation.

Parameters

- **frag_record** – dictionary containing the .dat file names and their ids. The dictionary is to be assigned to
- **simulation** (*'Molecule_Files' property of the MC*) –

Returns updated dictionary

Return type dictionary

make_system(*self*, *text_output*)
pysimm.cassandra.McSystem.make_system

Parses the checkpoint (.chk) file made by CASSANDRA and creates new molecules basing on the new coordinates information. Assumes that all atoms of a certain molecule are listed in .chk file together (molecule identifiers are not mixed).

Note: The logic of comparison of the xyz-like text record from the .chk file with the *System* object is most straightforward: It is the consecutive comparison of particle names and first letters (before the white space) in the text record. In this implementation order matters! For example, for CO2, if in the system atoms are ordered as C-O-O and in the text they are ordered as O-C-O fit will fail.

Parameters **text_output** (*str*) – text stream from the CASSANDRA .chk file containing the coordinates of newly inserted molecules

Returns object containing all newly inserted molecules

Return type *System*

__fit_atoms__(*self*, *molec*, *text_lines*)
pysimm.cassandra.McSystem.__fit_atoms__

Implements simple logic of comparison of the xyz-like text record with the `System` object. The comparison is based on the consecutive comparison of particle names and first letters (before the white space) in the text. In this implementation order matters! E.g. for CO2, if in the system atoms are ordered as C-O-O and in the text they are ordered like O-C-O fit will return False.

Returns flag whether the text record fit the molecule or not

Return type boolean

`class pysimm.cassandra.Cassandra(init_sst)`

Bases: `object`

pysimm.cassandra.Cassandra

Organizational object for running CASSANDRA simulation tasks. In current implementation it is able to run Canonical, Grand Canonical, and Isothermal-Isobaric Monte Carlo simulations (`GCMC`, `NVT`, and `NPT`, correspondingly).

Parameters

- **`system`** (`System`) – molecular updated during the simulations
- **`run_queue`** (`list`) – the list of scheduled tasks

`run(self)`

pysimm.cassandra.Cassandra.run

Method that triggers the simulations. Does two consecutive steps: **(1)** tries to write all files necessary for simulation (.dat, .inp, .mcf): **(2)** tries to invoke the CASSANDRA executable.

`add_simulation(self, ens_type, obj=None, **kwargs)`

pysimm.cassandra.Cassandra.add_simulation

Method for adding new Monte Carlo simulation to the run queue.

Parameters

- **`ens_type`** – Type of the molecular ensemble for the Monte-Carlo simulations. The supported options are: *GCMC* (Grand Canonical); *NVT* (canonical); *NPT* (isobaric-isothermal)
- **`obj`** – the entity that should be added. Will be ignored if it is not of a type `MCSimulation`

Keyword Arguments

- **`is_new`** (`boolean`) – defines whether all previous simulations should be erased or not
- **`species`** (list of `System`) – systems that describe molecules and will be passed to `McSystem` constructor.

Note: Other keyword arguments of this method will be redirected to the `McSystem` and `MCSimulation` constructors. See their descriptions for the possible keyword options.

`add_gcmc(self, obj=None, **kwargs)`

pysimm.cassandra.Cassandra.add_gcmc

Ads new simulation in grand-canonical ensemble to the run queue.

Parameters **`obj`** – the entity that should be added. Will be ignored if it is not of a type `GCMC`

Keyword Arguments

- **`is_new`** (`boolean`) – defines whether all previous simulations should be erased or not

- **species** (list of *System*) – systems that describe molecules and will be passed to *McSystem* constructor.

Note:

Other keyword arguments of this method will be redirected to the *McSystem*, *MCSimulation*, and *GCMC* constructors. See their descriptions for the possible keyword options.

---

add_npt_mc(*self*, *obj=None*, ***kwargs*)
pysimm.cassandra.Cassandra.add_npt_mc

Ads new simulation in isobaric-isothermal ensemble to the run queue.

Parameters **obj** – the entity that should be added. Will be ignored if it is not of a type *NPT*

Keyword Arguments

- **is_new** (*boolean*) – defines whether all previous simulations should be erased or not
- **species** (list of *System*) – systems that describe molecules and will be passed to *McSystem* constructor.

Note: Other keyword arguments of this method will be redirected to the *McSystem*, *MCSimulation*, and *NPT* constructors. See their descriptions for the possible keyword options.

---

add_nvt(*self*, *obj=None*, ***kwargs*)
pysimm.cassandra.Cassandra.add_nvt

Ads new simulation in canonical ensemble to the run queue.

Parameters **obj** – the entity that should be added. Will be ignored if it is not of a type *NVT*

Keyword Arguments

- **is_new** (*boolean*) – defines whether all previous simulations should be erased or not
- **species** (list of *System*) – systems that describe molecules and will be passed to *McSystem* constructor.

Note: Other keyword arguments of this method will be redirected to the *McSystem*, *MCSimulation*, and *NVT* constructors. See their descriptions for the possible keyword options.

---

read_input(*self*, *inp_file*)
pysimm.cassandra.Cassandra.read_input

The method parses the CASSANDRA instructions file (.inp) split it into separate instructions and analyses each according to the instruction name.

Parameters **inp_file** (*str*) – the full relative path of the file to be read

Returns read CASSANDRA properties in the format required by *GCMC*

Return type dictionary

__parse_value__(*self*, *cells*)

unwrap_gas(*self*)
pysimm.cassandra.Cassandra.unwrap_gas

Ensures that all particles that are not fixed are unwrapped, otherwise CASSANDRA might not interpret them correctly

class `pysimm.cassandra.`**`McfWriter`**(*syst*, *file_ref*)

Bases: `object`

pysimm.cassandra.McfWriter

Object responsible for creating the CASSANDRA Molecular Configuration file (.mcf).

`syst`

represents the molecule to be described

Type *`System`*

`file_ref`

full relative path to the file that will be created

Type str

`mcf_tags = ['# Bond_Info', '# Angle_Info', '# Dihedral_Info', '# Improper_Info', '# Intra_Scaling', '#...`

`empty_line = 0`

`write`(*self*, *typing='all'*)

pysimm.cassandra.McfWriter.write

Method creates the .mcf file writing only those sections of it that are marked to be written

Parameters **`typing`** (*`list`*) – the list of sections to be written or the text keyword. List items should be as they are defined in *`mcf_tags`* field); default 'all'

`__write_empty__`(*self*, *out*, *name*)

`__write_atom_info__`(*self*, *out*)

`__write_bond_info__`(*self*, *out*)

`__write_angle_info__`(*self*, *out*)

`__write_intra_scaling__`(*self*, *out*)

`__write_dihedral_info__`(*self*, *out*)

`__write_improper_info__`(*self*, *out*)

`__write_fragment_info__`(*self*, *out*)

`__write_fragment_connectivity__`(*self*, *out*)

`__to_tags__`(*self*, *inpt*)

`pysimm.cassandra.`**`check_cs_exec`**()

pysimm.cassandra.check_cs_exec

Validates that the absolute path to the CASSANDRA executable is set in the *CASSANDRA_EXEC* environmental variable of the OS. The validation is called once inside the *`run`* method.

`pysimm.cassandra.`**`make_iterable`**(*obj*)

pysimm.cassandra.make_iterable

Utility method that forces the attributes be iterable (wrap in a list if it contains of only one item)

`pysimm.cli`

Module Contents

`pysimm.cli.supported_forcefields = ['dreiding', 'pcff', 'gaff']`

`pysimm.cli.parser`

`pysimm.lmps`

Module Contents

Classes

Init	pysimm.lmps.Init
Region	pysimm.lmps.Region
CreateBox	pysimm.lmps.CreateBox
Group	pysimm.lmps.Group
Velocity	pysimm.lmps.Velocity
OutputSettings	pysimm.lmps.OutputSettings
Qeq	pysimm.lmps.MolecularDynamics
MolecularDynamics	pysimm.lmps.MolecularDynamics
SteeredMolecularDynamics	pysimm.lmps.MolecularDynamics
Minimization	pysimm.lmps.Minimization
CustomInput	pysimm.lmps.CustomInput
Simulation	pysimm.lmps.Simulation
LogFile	pysimm.lmps.LogFile

Functions

check_lmps_exec()	
enqueue_output(out, queue)	pysimm.lmps.enqueue_output
call_lammps(simulation, np, nanohub, prefix='mpiexec')	pysimm.lmps.call_lammps
qeq(s, np=None, nanohub=None, **kwargs)	pysimm.lmps.qeq
quick_md(s, np=None, nanohub=None, **kwargs)	pysimm.lmps.quick_md
quick_min(s, np=None, nanohub=None, **kwargs)	pysimm.lmps.quick_min
energy(s, all=False, np=None, **kwargs)	pysimm.lmps.energy
check_lmps_attr(s)	

Attributes

pd
LAMMPS_EXEC
verbose
templates
FF_SETTINGS

`pysimm.lmps.`**`pd`**

`pysimm.lmps.`**`LAMMPS_EXEC`**

`pysimm.lmps.`**`verbose = False`**

`pysimm.lmps.`**`templates`**

`pysimm.lmps.`**`FF_SETTINGS`**

`pysimm.lmps.`**`check_lmps_exec`**`()`

`class` `pysimm.lmps.`**`Init`**`(**`*kwargs*`)`

Bases: `object`

pysimm.lmps.Init

Template object to contain LAMMPS initialization settings

`forcefield`
name of a supported force field; simulation settings will be chosen based on the force field name

`units`
LAMMPS set of units to use during simulation; default=real

`atom_style`
LAMMPS aomt_style to use during simulation; default=full

`charge`
option to define if any particles in system a non-zero charge

`kspace_style`
LAMMPS kspace_style to use during simulation if system has charges; default=pppm 1e-4

`cutoff`
dictionary of cutoff distances for nonbonded interactions; default={'lj': 12.0, 'coul': 12.0, 'inner_lj': 10.0}

`pair_style`
LAMMPS pair_style to use during simulation

`bond_style`
LAMMPS bond_style to use during simulation

`angle_style`
LAMMPS angle_style to use during simulation

`dihedral_style`
LAMMPS dihedral_style to use during simulation

improper_style
LAMMPS improper_style to use during simulation

special_bonds
LAMMPS special_bonds to use during simulation

pair_modify
LAMMPS pair_modify to use during simulation

read_data
name of data file to read instead of using `System` object

write(*self*, *sim=None*)
pysimm.lmps.Init.write

Prepare LAMMPS input with initialization settings

Parameters **sim** – `Simulation` object reference

Returns string of LAMMPS input

class pysimm.lmps.**Region**(*name='all'*, *style='block'*, **args*, ***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.lmps.Region

Template object to create a region in a LAMMPS simulation. See LAMMPS documentation for further information

name
name id for region

style
LAMMPS region style

***args**
args for given style

****kwargs**
optional kwargs for region command

write(*self*, *sim=None*)

class pysimm.lmps.**CreateBox**(*n=1*, *region=Region()*, **args*, ***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.lmps.CreateBox

Template object to create a box in a LAMMPS simulation. See LAMMPS documentation for further information

n
number of atom types

region
`Region` object

****kwargs**
optional kwargs for create_box command (replace / with _)

write(*self*, *sim=None*)

class pysimm.lmps.**Group**(*name='all'*, *style='id'*, **args*, ***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.lmps.Group

Template object to define a group in a LAMMPS simulation. See LAMMPS documentation for further information

`name`
    name for the group

`style`
    style for the group

`\*args`
    arguments for the given style

`write`(*self*, *sim=None*)

`class pysimm.lmps.Velocity`(*group=Group('all')*, *style='create'*, **args*, ***kwargs*)
    Bases: `pysimm.utils.Item`

pysimm.lmps.Velocity

Template object to define velocity initialization in a LAMMPS simulation. See LAMMPS documentation for further information

`group`
    group for velocity command

`style`
    style for the velocity command

`\*args`
    arguments for the given style

`write`(*self*, *sim=None*)

`class pysimm.lmps.OutputSettings`(***kwargs*)
    Bases: `object`

pysimm.lmps.OutputSettings

Template object to define thermo and dump output settings in a LAMMPS simulation. See LAMMPS documentation for further information

`thermo`
    dictionary of settings for thermo output

`dump`
    dictionary of settings for dump output

`write`(*self*, *sim=None*)

`class pysimm.lmps.Qeq`(***kwargs*)
    Bases: `object`

pysimm.lmps.MolecularDynamics

Template object to contain LAMMPS qeq settings

`cutoff`
    distance cutoff for charge equilibration

`tol`
    tolerance (precision) for charge equilibration

`max_iter`
    maximum iterations

`qfile`
: file with qeq parameters (leave undefined for defaults)

`write`(*self*, *sim=None*)
: pysimm.lmps.Qeq.write

  Create LAMMPS input for a charge equilibration calculation

  Parameters `sim` – *Simulation* object reference

  Returns input string

`class pysimm.lmps.`**`MolecularDynamics`**`(`*****kwargs***`)`
: Bases: `object`

  pysimm.lmps.MolecularDynamics

  Template object to contain LAMMPS MD settings

  `name`
  : name to identify MD

  `group`
  : *Group* object for integrator

  `timestep`
  : timestep value to use during MD

  `ensemble`
  : 'nvt' or 'npt' or 'nve'; default=nve

  `limit`
  : numerical value to use with nve when limiting particle displacement

  `temperature`
  : dictionary of settings for temperature (start, stop, damp)

  `pressure`
  : dictionary of settings for pressure (start, stop, damp)

  `run`
  : length of MD simulation in number of timesteps or False to omit run command

  `unfix`
  : True to include command to unfix integrator after run

  `rigid`
  : dictionary of settings for a rigid simulation

  `extra_keywords`
  : dictionary of extra keywords to append at the end of the LAMMPS fix integrator

  `write`(*self*, *sim=None*)
  : pysimm.lmps.MolecularDynamics.write

    Create LAMMPS input for a molecular dynamics simulation.

    Parameters `sim` – pysimm.lmps.Simulation object reference

    Returns input string

`class pysimm.lmps.`**`SteeredMolecularDynamics`**`(`*****kwargs***`)`
: Bases: *MolecularDynamics*

  pysimm.lmps.MolecularDynamics

Template object to contain LAMMPS MD settings

`name`
: name to identify MD

`group`
: *Group* object for integrator

`timestep`
: timestep value to use during MD

`ensemble`
: 'nvt' or 'npt' or 'nve'; default=nve

`limit`
: numerical value to use with nve when limiting particle displacement

`temperature`
: dictionary of settings for temperature (start, stop, damp)

`pressure`
: dictionary of settings for pressure (start, stop, damp)

`run`
: length of MD simulation in number of timesteps or False to omit run command

`unfix`
: True to include command to unfix integrator after run

`rigid`
: dictionary of settings for a rigid simulation

`extra_keywords`
: dictionary of extra keywords to append at the end of the LAMMPS fix integrator

`write`(*self*, *sim=None*)

pysimm.lmps.SteeredMolecularDynamics.write

Create LAMMPS input for a steered molecular dynamics simulation.

Parameters **`sim`** – *Simulation* object reference

Returns input string

class pysimm.lmps.**Minimization**(***kwargs*)

Bases: `object`

pysimm.lmps.Minimization

Template object to contain LAMMPS energy minimization settings.

`min_style`
: LAMMPS minimization style default='sd'

`dmax`
: how far any atom can move in a single line search in any dimension

`etol`
: energy tolerance default=1e-3

`ftol`
: force tolerance default=1e-3

`maxiter`
: maximum iterations default=10000

`max eval`
maximum force evaluations default=100000

`write`(*self*, *sim=None*)
pysimm.lmps.Minimization.write

Create LAMMPS input for an energy minimization simulation.

Parameters `sim` – *`Simulation`* object reference

Returns input string

`class pysimm.lmps.CustomInput`(*custom_input*)
Bases: `object`

pysimm.lmps.CustomInput

Template object to contain custom LAMMPS input.

`custom_input`
custom input string

`write`(*self*, *sim=None*)
pysimm.lmps.CustomInput.write

Create LAMMPS input for a custom simulation.

Parameters `sim` – pysimm.lmps.Simulation object reference

Returns input string

`class pysimm.lmps.Simulation`(*s*, ***kwargs*)
Bases: `object`

pysimm.lmps.Simulation

Organizational object for LAMMPS simulation. Should contain combination of *`MolecularDynamics`*, *`Minimization`*, and/or *`CustomInput`* object.

`forcefield`
name of force field for simulation settings

`name`
name for simulation

`log`
LAMMPS log filename

`write`
file name to write final LAMMPS data file default=None

`print_to_screen`
True to have LAMMPS output printed to stdout after simulation ends

`debug`
True to have LAMMPS output streamed to stdout during simulation (WARNING: this may degrade performance)

`custom`
option to flag simulation as purley custom input to skip prepaing initialization

`add`(*self*, **args*)

`add_qeq`(*self*, *template=None*, ***kwargs*)
pysimm.lmps.Simulation.add_qeq

Add `Qeq` template to simulation

Parameters

- **`template`** – `Qeq` object reference
- **`**kwargs`** – if template is None these are passed to `Qeq` constructor to create new template

`add_md`(*self*, *template=None*, ***kwargs*)
pysimm.lmps.Simulation.add_md

Add `MolecularDyanmics` template to simulation

Parameters

- **`template`** – `MolecularDynamics` object reference
- **`**kwargs`** – if template is None these are passed to `MolecularDynamics` constructor to create new template

`add_min`(*self*, *template=None*, ***kwargs*)
pysimm.lmps.Simulation.add_min

Add `Minimization` template to simulation

Parameters

- **`template`** – `Minimization` object reference
- **`**kwargs`** – if template is None these are passed to `Minimization` constructor to create new template

`add_custom`(*self*, *custom=''*)
pysimm.lmps.Simulation.add_custom

Add custom input string to simulation

Parameters **`custom`** – custom LAMMPS input string to add to Simulation

`property input`(*self*)

`write_input`(*self*, *init=True*)
pysimm.lmps.Simulation.write_input

Creates LAMMPS input string including initialization and input from templates/custom input

Parameters **`None`** –

Returns None

`run`(*self*, *np=None*, *nanohub=None*, *save_input=True*, *prefix='mpiexec'*)
pysimm.lmps.Simulation.run

Begin LAMMPS simulation.

Parameters

- **`np`** – number of threads to use (serial by default) default=None
- **`nanohub`** – dictionary containing nanohub resource information default=None
- **`init`** – True to write initialization part of LAMMPS input script (set to False if using complete custom input)
- **`save_input`** – True to save input as pysimm.sim.in
- **`prefix`** – prefix for running LAMMPS (i.e. - mpiexec)

`pysimm.lmps.`**`enqueue_output`**`(`*out*, *queue*`)`
pysimm.lmps.enqueue_output

Helps queue output for printing to screen during simulation.

`pysimm.lmps.`**`call_lammps`**`(`*simulation*, *np*, *nanohub*, *prefix='mpiexec'*`)`
pysimm.lmps.call_lammps

Wrapper to call LAMMPS using executable name defined in pysimm.lmps module.

Parameters

- **`simulation`** – *`Simulation`* object reference
- **`np`** – number of threads to use
- **`nanohub`** – dictionary containing nanohub resource information default=None
- **`prefix`** – prefix for running LAMMPS (i.e. - mpiexec)

Returns None

`pysimm.lmps.`**`qeq`**`(`*s*, *np=None*, *nanohub=None*, ***kwargs*`)`
pysimm.lmps.qeq

Convenience function to call a qeq calculation. kwargs are passed to *`Qeq`* constructor

Parameters

- **`s`** – system to perform simulation on
- **`np`** – number of threads to use
- **`nanohub`** – dictionary containing nanohub resource information default=None

Returns None

`pysimm.lmps.`**`quick_md`**`(`*s*, *np=None*, *nanohub=None*, ***kwargs*`)`
pysimm.lmps.quick_md

Convenience function to call an individual MD simulation. kwargs are passed to MD constructor

Parameters

- **`s`** – system to perform simulation on
- **`np`** – number of threads to use
- **`nanohub`** – dictionary containing nanohub resource information default=None

Returns None

`pysimm.lmps.`**`quick_min`**`(`*s*, *np=None*, *nanohub=None*, ***kwargs*`)`
pysimm.lmps.quick_min

Convenience function to call an individual energy minimization simulation. kwargs are passed to min constructor

Parameters

- **`s`** – system to perform simulation on
- **`np`** – number of threads to use
- **`nanohub`** – dictionary containing nanohub resource information default=None

Returns None

`pysimm.lmps.energy(s, all=False, np=None, **kwargs)`

pysimm.lmps.energy

Convenience function to calculate energy of a given `System` object.

Parameters

- **s** – system to calculate energy
- **all** – returns decomposition of energy if True (default: False)
- **np** – number of threads to use for simulation

Returns total energy or disctionary of energy components

`pysimm.lmps.check_lmps_attr(s)`

`class pysimm.lmps.LogFile(fname)`

Bases: `object`

pysimm.lmps.LogFile

Class to read LAMMPS log file into Pandas DataFrame stored in LogFile.data

fname
filename of log file

data
resulting DataFrame with log file data

`_read(self, fname)`

pysimm.system

Module Contents

Classes

`Particle`	pysimm.system.Particle
`ParticleType`	pysimm.system.ParticleType
`Bond`	pysimm.system.Bond
`BondType`	pysimm.system.BondType
`Angle`	pysimm.system.Angle
`AngleType`	pysimm.system.AngleType
`Dihedral`	pysimm.system.Dihedral
`DihedralType`	pysimm.system.DihedralType
`Improper`	pysimm.system.Improper
`ImproperType`	pysimm.system.ImproperType
`Dimension`	pysimm.system.Dimension
`System`	pysimm.system.System
`Molecule`	pysimm.system.Molecule

Functions

read_yaml(file_, **kwargs)	pysimm.system.read_yaml
read_xyz(file_, **kwargs)	pysimm.system.read_xyz
read_chemdoodle_json(file_, **kwargs)	pysimm.system.read_chemdoodle_json
read_lammps(data_file, **kwargs)	pysimm.system.read_lammps
read_pubchem_smiles(smiles, quiet=False, type_with=None)	pysimm.system.read_pubchem_smiles
read_pubchem_cid(cid, type_with=None)	pysimm.system.read_pubchem_smiles
read_cml(cml_file, **kwargs)	pysimm.system.read_cml
read_mol(mol_file, type_with=None, version='V2000')	pysimm.system.read_mol
read_mol2(mol2_file, type_with=None)	pysimm.system.read_mol2
read_prepc(prec_file)	pysimm.system.read_prepc
read_ac(ac_file)	pysimm.system.read_ac
read_pdb(pdb_file, str_file=None, **kwargs)	pysimm.system.read_pdb
compare(s1, s2)	
get_types(*arg, **kwargs)	pysimm.system.get_types
distance_to_origin(p)	pysimm.system.distance_to_origin
replicate(ref, nrep, s_=None, density=0.3, rand=True, print_insertions=True)	pysimm.system.replicate

Attributes

call
np
pd

pysimm.system.**call**

pysimm.system.**np**

pysimm.system.**pd**

class pysimm.system.**Particle**(***kwargs*)

Bases: *pysimm.utils.Item*

pysimm.system.Particle

Objects inheriting from *Item* can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

x

x coordinate

y

y coordinate

z

z coordinate

charge
partial charge

type
ParticleType object reference

coords(*self*)

check(*self*, *style='full'*)

delete_bonding(*self*, *s*)
pysimm.system.Particle.delete_bonding

Iterates through s.bonds, s.angles, s.dihedrals, and s.impropers and removes those which contain this *Particle*.

Parameters **s** – *System* object from which bonding objects will be removed

Returns None

translate(*self*, *dx*, *dy*, *dz*)
pysimm.system.Particle.translate

Shifts Particle position by dx, dy, dz.

Parameters

- **dx** – distance to shift in x direction
- **dy** – distance to shift in y direction
- **dz** – distance to shift in z direction

Returns None

__sub__(*self*, *other*)
pysimm.system.Particle.__sub__

Implements subtraction between *Particle* objects to calculate distance.

Parameters **other** – *Particle* object

Returns distance calculated by *distance()*. This does not consider pbc

__rsub__(*self*, *other*)

class pysimm.system.**ParticleType**(***kwargs*)
Bases: *pysimm.utils.Item*

pysimm.system.ParticleType

Objects inheriting from *Item* can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

sigma
LJ sigma value (Angstrom)

epsilon
LJ epsilon value (kcal/mol)

elem
element abbreviation, i.e. 'H' for Hydrogen, 'Cl' for Chlorine

name
force field particle type name

form(*self*, *style='lj_12-6'*, *d_range=None*)
pysimm.system.ParticleType.form

Returns data to plot functional form for the potential energy with the given style.

Parameters **style** – string for pair style of ParticleType (lj_12-6, lj_9-6, buck)

Returns x, y for plotting functional form (energy vs distance)

classmethod guess_style(*cls*, *nparam*)

classmethod parse_lammps(*cls*, *line*, *style*)

write_lammps(*self*, *style='lj'*)
pysimm.system.ParticleType.write_lammps

Formats a string to define particle type coefficients for a LAMMPS data file given the provided style.

Parameters **style** – string for pair style of ParticleType (lj, class2, mass, buck)

Returns LAMMPS formatted string with pair coefficients

class pysimm.system.**Bond**(***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.system.Bond

Bond between particle a and b

a–b

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

a
`Particle` object involved in bond

b
`Particle` object involved in bond

type
BondType object reference

get_other_particle(*self*, *p*)

distance(*self*)
pysimm.system.Bond.distance

Calculates distance between `Particle` a and `Particle` b in this Bond object. Sets distance to dist attribute of self. Does not consider pbc.

Parameters **None** –

Returns Distance between Particle a and Particle b (not considering pbc)

class pysimm.system.**BondType**(***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.system.BondType

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

k
harmonic bond force constant (kcal/mol/A^2)

`r0`
bond equilibrium distance (Angstrom)

`name`
force field bond type name

`classmethod guess_style`(*cls*, *nparam*)

`classmethod parse_lammps`(*cls*, *line*, *style*)

`write_lammps`(*self*, *style='harmonic'*)
pysimm.system.BondType.write_lammps

Formats a string to define bond type coefficients for a LAMMPS data file given the provided style.

Parameters `style` – string for pair style of BondType (harmonic, class2)

Returns LAMMPS formatted string with bond coefficients

`form`(*self*, *style='harmonic'*, *d_range=None*)
pysimm.system.BondType.form

Returns data to plot functional form for the potential energy with the given style.

Parameters `style` – string for pair style of BondType (harmonic, class2)

Returns x, y for plotting functional form (energy vs distance)

`class pysimm.system.Angle`(***kwargs*)
Bases: *pysimm.utils.Item*

pysimm.system.Angle

Angle between particles a, b, and c

a–b–c

Objects inheriting from *Item* can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

`a`
Particle object involved in angle

`b`
Particle object involved in angle (middle particle)

`c`
Particle object involved in angle

`type`
AngleType object reference

`angle`(*self*, *radians=False*)
pysimm.system.Angle.angle

Calculate angle.

Parameters `radians` – True to return value in radians (default: False)

Returns Angle between Particle a, b, and c

`class pysimm.system.AngleType`(***kwargs*)
Bases: *pysimm.utils.Item*

pysimm.system.AngleType

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

k
harmonic angle bend force constant (kcal/mol/radian^2)

theta0
angle equilibrium value (degrees)

name
force field angle type name

classmethod guess_style(*cls*, *nparam*)

classmethod parse_lammps(*cls*, *line*, *style*)

write_lammps(*self*, *style='harmonic'*, *cross_term=None*)
pysimm.system.AngleType.write_lammps

Formats a string to define angle type coefficients for a LAMMPS data file given the provided style.

Parameters

- **style** – string for pair style of AngleType (harmonic, class2, charmm)
- **cross_term** – type of class2 cross term to write (default=None) - BondBond - BondAngle

Returns LAMMPS formatted string with angle coefficients

form(*self*, *style='harmonic'*, *d_range=None*)
pysimm.system.AngleType.form

Returns data to plot functional form for the potential energy with the given style.

Parameters **style** – string for pair style of AngleType (harmonic, class2, charmm)

Returns x, y for plotting functional form (energy vs angle)

class pysimm.system.**Dihedral**(***kwargs*)
Bases: `pysimm.utils.Item`

pysimm.system.Dihedral

Dihedral between particles a, b, c, and d

a–b–c–d

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

a
`Particle` object involved in dihedral

b
`Particle` object involved in dihedral (middle particle)

c
`Particle` object involved in dihedral (middle particle)

d
`Particle` object involved in dihedral

type
`DihedralType` object reference

class pysimm.system.**DihedralType**(***kwargs*)

Bases: `pysimm.utils.Item`

pysimm.system.DihedralType

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

k
: dihedral energy barrier (kcal/mol)

d
: minimum (+1 or -1)

n
: multiplicity (integer >= 0)

name
: force field dihedral type name

classmethod guess_style(*cls*, *nparam*)

classmethod parse_lammps(*cls*, *line*, *style*)

write_lammps(*self*, *style='harmonic'*, *cross_term=None*)

pysimm.system.DihedralType.write_lammps

Formats a string to define dihedral type coefficients for a LAMMPS data file given the provided style.

Parameters

- **style** – string for pair style of DihedralType (harmonic, class2, fourier)
- **cross_term** – type of class2 cross term to write (default=None) - MiddleBond - EndBond - Angle - AngleAngle - BondBond13

Returns LAMMPS formatted string with dihedral coefficients

form(*self*, *style='harmonic'*, *d_range=None*)

pysimm.system.DihedralType.form

Returns data to plot functional form for the potential energy with the given style.

Parameters **style** – string for pair style of DihedralType (harmonic, class2, fourier)

Returns x, y for plotting functional form (energy vs angle)

class pysimm.system.**Improper**(***kwargs*)

Bases: `pysimm.utils.Item`

pysimm.system.Improper

Improper dihedral around particle a, bonded to b, c, and d

b
|
a–d
|
c

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

`a`
: `Particle` object involved in improper (middle particle)

`b`
: `Particle` object involved in improper

`c`
: `Particle` object involved in improper

`d`
: `Particle` object involved in improper

`type`
: `ImproperType` object reference

`class pysimm.system.`**`ImproperType`**`(`*`**kwargs`*`)`

Bases: `pysimm.utils.Item`

pysimm.system.ImproperType

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

`k`
: improper energy barrier (kcal/mol)

`x0`
: equilibrium value (degrees)

`name`
: force field improper type name

`classmethod guess_style``(`*cls*, *nparam*`)`

`classmethod parse_lammps``(`*cls*, *line*, *style*`)`

`write_lammps``(`*self*, *style='harmonic'*, *cross_term=None*`)`

pysimm.system.ImproperType.write_lammps

Formats a string to define improper type coefficients for a LAMMPS data file given the provided style.

Parameters

- **`style`** – string for pair style of ImproperType (harmonic, class2, cvff)
- **`cross_term`** – type of class2 cross term to write (default=None) - AngleAngle

Returns LAMMPS formatted string with dihedral coefficients

`form``(`*self*, *style='harmonic'*, *d_range=None*`)`

pysimm.system.ImproperType.form

Returns data to plot functional form for the potential energy with the given style.

Parameters **`style`** – string for pair style of ImproperType (harmonic, cvff)

Returns x, y for plotting functional form (energy vs angle)

`class pysimm.system.`**`Dimension`**`(`*`**kwargs`*`)`

Bases: `pysimm.utils.Item`

pysimm.system.Dimension

Objects inheriting from `Item` can contain arbitrary data. Keyword arguments are assigned as attributes. Attributes usually used are given below.

xlo
minimum value in x dimension

xhi
maximum value in x dimension

ylo
minimum value in y dimension

yhi
maximum value in y dimension

zlo
minimum value in z dimension

zhi
maximum value in z dimension

dx
distance in x dimension

dy
distance in y dimension

dz
distance in z dimension

check(*self*)

size(*self*)

translate(*self*, *x*, *y*, *z*)
pysimm.system.Dimension.translate

Shifts box bounds by x, y, z.

Parameters

- **x** – distance to shift box bounds in x direction
- **y** – distance to shift box bounds in y direction
- **z** – distance to shift box bounds in z direction

Returns None

property **dx**(*self*)

property **dy**(*self*)

property **dz**(*self*)

class pysimm.system.**System**(***kwargs*)
Bases: `object`

pysimm.system.System

Object representation of molecular system. Contains information required for molecular simulation.

dim
Dimension object reference

particles
ItemContainer for Particle organization

particle_types
ItemContainer for ParticleType organization

bonds
: *ItemContainer* for Bond organization

bond_types
: *ItemContainer* for BondType organization

angles
: *ItemContainer* for Angle organization

angle_types
: *ItemContainer* for AngleType organization

dihedrals
: *ItemContainer* for Dihedral organization

dihedral_types
: *ItemContainer* for DihedralType organization

impropers
: *ItemContainer* for Improper organization

improper_types
: *ItemContainer* for ImproperType organization

molecules
: *ItemContainer* for Molecule organization

__getattr__(*self*, *name*)

copy(*self*, *rotate_x=None*, *rotate_y=None*, *rotate_z=None*, *dx=0*, *dy=0*, *dz=0*)
pysimm.system.System.copy

Create duplicate *System* object. Default behavior does not modify particle positions.

Parameters

- **rotate_x** – rotate duplicate system around x axis by this value (radians)
- **rotate_y** – rotate duplicate system around y axis by this value (radians)
- **rotate_z** – rotate duplicate system around z axis by this value (radians)
- **dx** – translate duplicate system in x dimension by this value (Angstrom)
- **dy** – translate duplicate system in y dimension by this value (Angstrom)
- **dz** – translate duplicate system in z dimension by this value (Angstrom)

add(*self*, *other*, ***kwargs*)
pysimm.system.System.add

Add other *System* to this. Optionally remove duplicate types (default behavior).

Parameters

- **other** – *System* object to add
- **unique_types** (*optional*) – Remove duplicate types and reassign references to existing types (True)
- **change_dim** (*optional*) – Update *Dimension* object so that *Particle* objects do not exist outside of *Dimension* extremes (True)
- **update_properties** (*optional*) – Update system-wide mass, volume, density, center of gravity, and velocity properties (True)

`distance`(*self*, *p1*, *p2*)

pysimm.system.System.distance

Calculate distance between two particles considering pbc.

Parameters

- **`p1`** – *`Particle`* object
- **`p2`** – *`Particle`* object

Returns distance between particles considering pbc

`wrap`(*self*)

pysimm.system.System.wrap

Wrap *`Particle`* images into box defined by *`Dimension`* object. Ensure particles are contained within simulation box.

Parameters **`None`** –

Returns None

`unwrap`(*self*)

pysimm.system.System.unwrap()

Unwraps *`Particle`* images such that no bonds cross box edges.

Parameters **`None`** –

Returns None

`particles_df`(*self*, *columns=['tag', 'x', 'y', 'z', 'q']*, *index='tag'*, *extras=[]*)

`unite_atoms`(*self*)

`quality`(*self*, *tolerance=0.1*)

pysimm.system.System.quality

Attemps to assess quality of *`System`* based on bond lengths in unwrapped system.

Parameters **`tolerance`** – fractional value of equilibrium bond length that is acceptable

Returns number of bonds in system outside tolerance

`shift_to_origin`(*self*)

pysimm.system.System.shift_to_origin

Shifts simulation box to begin at origin. i.e. xlo=ylo=zlo=0

Parameters **`None`** –

Returns None

`set_charge`(*self*)

pysimm.system.System.set_charge

Sets total charge of all *`Particle`* objects in System.particles

Parameters **`None`** –

Returns None

`zero_charge`(*self*)

pysimm.system.System.zero_charge

Enforces total *`System`* charge to be 0.0 by subtracting excess charge from last particle

Parameters **`None`** –

Returns None

`check_items`(*self*)
pysimm.system.System.check_items

Checks particles, bonds, angles, dihedrals, impropers, and molecules containers and raises exception if the length of items in the container does not equal the count property

Parameters **`None`** –

Returns None

`update_ff_types_from_ac`(*self*, *ff*, *acname*)
pysimm.system.System.update_ff_types_from_ac

Updates *`ParticleType`* objects in system using type names given in antechamber (ac) file. Retrieves type from System if possible, then searches force field provided by ff.

Parameters

- **`ff`** – forcefield to search for Type objects
- **`acname`** – ac filename containing type names

Returns None

`update_particle_types_from_forcefield`(*self*, *f*)
pysimm.system.System.update_types_from_forcefield

Updates *`ParticleType`* data from *`Forcefield`* object f based on *`ParticleType`*.name

Parameters **`f`** – *`Forcefield`* object reference

Returns None

`make_linker_types`(*self*)
pysimm.system.System.make_linker_types

Identifies linker particles and creates duplicate `Particle`.linker attribute. New *`ParticleType`* name is prepended with [H or T]L@ to designate head or tail linker

Parameters **`None`** –

Returns None

`remove_linker_types`(*self*)
pysimm.system.System.remove_linker_types

Reassigns *`Particle`*.type references to original *`ParticleType`* objects without linker prepend

Parameters **`None`** –

Returns None

`read_lammps_dump`(*self*, *fname*)
pysimm.system.System.read_lammps_dump

Updates particle positions and box size from LAMMPS dump file. Assumes following format for each atom line:

tag charge xcoord ycoord zcoord xvelocity yvelocity zvelocity

Parameters **`fname`** – LAMMPS dump file

Returns None

`read_lammpstrj`(*self*, *trj*, *frame=1*)
pysimm.system.System.read_lammpstrj

Updates particle positions and box size from LAMMPS trajectory file at given frame.

Assumes one of following formats for each atom line:

tag xcoord ycoord zcoord

OR

tag type_id xcoord ycoord zcoord

OR

tag type_id xcoord ycoord zcoord ximage yimage zimage

Parameters

- **`trj`** – LAMMPS trajectory file
- **`frame`** – sequential frame number (not LAMMPS timestep) default=1

Returns None

`read_xyz`(*self*, *xyz*, *frame=1*, *quiet=False*)
pysimm.system.System.read_xyz

Updates particle positions and box size from xyz file at given frame

Parameters

- **`xyz`** – xyz trajectory file
- **`frame`** – sequential frame number default=1
- **`quiet`** – True to print status default=False

Returns None

`update_types`(*self*, *ptypes*, *btypes*, *atypes*, *dtypes*, *itypes*)
pysimm.system.System.update_types

Updates type objects from a given list of types.

Parameters

- **`ptypes`** – list of `ParticleType` objects from which to update
- **`btypes`** – list of `BondType` objects from which to update
- **`atypes`** – list of `AngleType` objects from which to update
- **`dtypes`** – list of `DihedralType` objects from which to update
- **`itypes`** – list of `ImproperType` objects from which to update

`read_type_names`(*self*, *types_file*)
pysimm.system.System.read_type_names

Update `ParticleType` names from file.

Parameters **`types_file`** – type dictionary file name

Returns None

`remove_spare_bonding`(*self*, *update_tags=True*)
pysimm.system.System.remove_spare_bonding

Removes bonds, angles, dihedrals and impropers that reference particles not in `System`.particles

Parameters **`update_tags`** – True to update all tags after removal of bonding items default=True

`update_tags`(*self*)
pysimm.system.System.update_tags

Update Item tags in *`ItemContainer`* objects to preserve continuous tags. Removes all objects and then reinserts them.

Args: None

Returns: None

`set_references`(*self*)
pysimm.system.System.set_references

Set object references when *`System`* information read from text file. For example, if bond type value 2 is read from file, set *`Bond`*.type to bond_types[2]

Parameters **`None`** –

Returns None

`objectify`(*self*)
pysimm.system.System.objectify

Set references for *`Bond`*, *`Angle`*, *`Dihedral`*, *`Improper`* objects. For example, if read from file that bond #1 is between particle 1 and 2 set *`Bond`*.a to particles[1], etc.

Parameters **`None`** –

Returns None

`add_particle_bonding`(*self*)
pysimm.system.System.add_particle_bonding

Update *`Particle`* objects such that *`Particle`*.bonded_to contains other *`Particle`* objects invloved in bonding

Parameters **`None`** –

Returns None

`set_excluded_particles`(*self*, *bonds=True*, *angles=True*, *dihedrals=True*)
pysimm.system.System.set_excluded_particles

Updates *`Particle`* object such that *`Particle`*.excluded_particles contains other *`Particle`* objects involved in 1-2, 1-3, and/or 1-4 interactions

Parameters

- **`bonds`** – exclude particles involved in 1-2 interactions
- **`angles`** – exclude particles involved in 1-3 interactions
- **`dihedrals`** – exclude particles involved in 1-4 interactions

`set_atomic_numbers`(*self*)
pysimm.system.System.set_atomic_numbers

Updates *`ParticleType`* objects with atomic number based on *`ParticleType`*.elem

Parameters **`None`** –

Returns None

`add_particle_bonded_to`(*self*, *p*, *p0*, *f=None*, *sep=1.5*)
pysimm.system.System.add_particle_bonded_to

Add new `Particle` to `System` bonded to p0 and automatically update new forcefield types

Parameters

- `p` – new `Particle` object to be added
- `p0` – original `Particle` object in `System` to which p will be bonded
- `f` – `Forcefield` object from which new force field types will be retrieved

Returns new Particle being added to system for convenient reference

`add_particle`(*self*, *p*)
pysimm.system.System.add_particle

Add new `Particle` to `System`.

Parameters `p` – new `Particle` object to be added

Returns None

`rotate`(*self*, *around=None*, *theta_x=0*, *theta_y=0*, *theta_z=0*, *rot_matrix=None*)
pysimm.system.System.rotate

*** REQUIRES NUMPY ***

Rotates entire system around given `Particle` by user defined angles

Parameters

- `around` – `Particle` around which `System` will be rotated default=None
- `theta_x` – angle around which system will be rotated on x axis
- `theta_y` – angle around which system will be rotated on y axis
- `theta_z` – angle around which system will be rotated on z axis
- `rot_matrix` – rotation matrix to use for rotation

Returns None

`make_new_bonds`(*self*, *p1=None*, *p2=None*, *f=None*, *angles=True*, *dihedrals=True*, *impropers=True*)
pysimm.system.System.make_new_bonds

Makes new bond between two particles and updates new force field types

Parameters

- `p1` – `Particle` object involved in new bond
- `p2` – `Particle` object involved in new bond
- `f` – `Forcefield` object from which new force field types will be retrieved
- `angles` – True to update new angles default=True
- `dihedrals` – True to update new dihedrals default=True
- `impropers` – True to update new impropers default=True

Returns None

`add_bond`(*self*, *a=None*, *b=None*, *f=None*)
pysimm.system.System.add_bond

Add `Bond` to system between two particles

Parameters

- **a** – *Particle* involved in new *Bond*
- **b** – *Particle* involved in new *Bond*
- **f** – *Forcefield* object from which new force field type will be retrieved

Returns None

add_angle(*self*, *a=None*, *b=None*, *c=None*, *f=None*)
pysimm.system.System.add_angle

Add *Angle* to system between three particles

Parameters

- **a** – *Particle* involved in new *Angle*
- **b** – *Particle* involved in new *Angle* (middle particle)
- **c** – *Particle* involved in new *Angle*
- **f** – *Forcefield* object from which new force field type will be retrieved

Returns None

add_dihedral(*self*, *a=None*, *b=None*, *c=None*, *d=None*, *f=None*)
pysimm.system.System.add_dihedral

Add *Dihedral* to system between four particles

Parameters

- **a** – *Particle* involved in new *Dihedral*
- **b** – *Particle* involved in new *Dihedral* (middle particle)
- **c** – *Particle* involved in new *Dihedral* (middle particle)
- **d** – *Particle* involved in new *Dihedral*
- **f** – *Forcefield* object from which new force field type will be retrieved

Returns None

add_improper(*self*, *a=None*, *b=None*, *c=None*, *d=None*, *f=None*)
pysimm.system.System.add_improper

Add *Improper* to system between four particles

Parameters

- **a** – `Particle` involved in new *Improper* (middle particle)
- **b** – `Particle` involved in new *Improper*
- **c** – `Particle` involved in new *Improper*
- **d** – `Particle` involved in new *Improper*
- **f** – `Forcefield` object from which new force field type will be retrieved

Returns None

check_forcefield(*self*)
pysimm.system.System.check_forcefield

Iterates through particles and prints the following:

tag type name type element type description bonded elements

Parameters `None` –

Returns None

`apply_forcefield`(*self*, *f*, *charges='default'*, *set_box=True*, *box_padding=10*, *update_ptypes=False*, *skip_ptypes=False*)
pysimm.system.System.apply_forcefield

Applies force field data to *System* based on typing rules defined in *Forcefield* object f

Parameters

- **f** – *Forcefield* object from which new force field type will be retrieved
- **charges** – type of charges to be applied default='default'
- **set_box** – Update simulation box information based on particle positions default=True
- **box_padding** – Add padding to simulation box if updating dimensions default=10 (Angstroms)
- **update_ptypes** – If True, update particle types based on current *ParticleType* names default=False
- **skip_ptypes** – if True, do not change particle types

Returns None

`apply_charges`(*self*, *f*, *charges='default'*)
pysimm.system.System.apply_charges

Applies charges derived using method provided by user. Defaults to 'default'. Calls `assign_charges()` method of forcefield object provided.

Parameters

- **f** – *Forcefield* object
- **charges** – type of charges to be applied default='default'

Returns None

`write_lammps_mol`(*self*, *out_data*)
pysimm.system.System.write_lammps_mol

Write *System* data formatted as LAMMPS molecule template

Parameters **out_data** – where to write data, file name or 'string'

Returns None or string if data file if out_data='string'

`write_lammps`(*self*, *out_data*, ***kwargs*)
pysimm.system.System.write_lammps

Write *System* data formatted for LAMMPS

Parameters **out_data** – where to write data, file name or 'string'

Returns None or string if data file if out_data='string'

`write_xyz`(*self*, *outfile='data.xyz'*, ***kwargs*)
pysimm.system.System.write_xyz

Write *System* data in xyz format

Parameters **outfile** – where to write data, file name or 'string'

Returns None or string of data file if out_data='string'

`write_chemdoodle_json`(*self*, *outfile*, ***kwargs*)
pysimm.system.System.write_chemdoodle_json

Write *System* data in chemdoodle json format

Parameters `outfile` – where to write data, file name or 'string'

Returns None or string of data file if out_data='string'

`write_mol`(*self*, *outfile='data.mol'*)
pysimm.system.System.write_mol

Write *System* data in mol format

Parameters `outfile` – where to write data, file name or 'string'

Returns None or string of data file if out_data='string'

`write_pdb`(*self*, *outfile='data.pdb'*, *type_names=True*)
pysimm.system.System.write_pdb

Write *System* data in pdb format

Parameters `outfile` – where to write data, file name or 'string'

Returns None or string of data file if out_data='string'

`write_yaml`(*self*, *file_*)
pysimm.system.System.write_yaml

Write *System* data in yaml format

Parameters `outfile` – file name to write data

Returns None

`write_cssr`(*self*, *outfile='data.cssr'*, ***kwargs*)
pysimm.system.System.write_cssr

Write *System* data in cssr format file format: line, format, contents 1: 38X, 3F8.3 : - length of the three cell parameters (a, b, and c) in angstroms. 2: 21X, 3F8.3, 4X, 'SPGR =', I3, 1X, A11 : - a, b, g in degrees, space group number, space group name. 3: 2I4, 1X, A60 : - Number of atoms stored, coordinate system flag (0=fractional, 1=orthogonal coordinates in Angstrom), first title. 4: A53 : - A line of text that can be used to describe the file. 5-: I4, 1X, A4, 2X, 3(F9.5.1X), 8I4, 1X, F7.3 : - Atom serial number, atom name, x, y, z coordinates, bonding connectivities (max 8), charge. Note: The atom name is a concatenation of the element symbol and the atom serial number.

Parameters

- `outfile` – where to write data, file name or 'string'
- `frac` – 0 for using fractional coordinates
- `aname` – 0 for using element as atom name; else using atom type name

Returns None or string of data file if out_data='string'

`consolidate_types`(*self*)
pysimm.system.System.consolidate_types

Removes duplicate types and reassigns references

Parameters `None` –

Returns None

`set_cog`(*self*)
pysimm.system.System.set_cog

Calculate center of gravity of *System* and assign to *System*.cog

Parameters `None` –

Returns None

`shift_particles`(*self*, *shiftx*, *shifty*, *shiftz*)
pysimm.system.System.shift_particles

Shifts all particles by shiftx, shifty, shiftz. Recalculates cog.

Parameters

- **`shiftx`** – distance to shift particles in x direction
- **`shifty`** – distance to shift particles in y direction
- **`shiftz`** – distance to shift particles in z direction

Returns None

`center`(*self*, *what='particles'*, *at=[0, 0, 0]*, *move_both=True*)
pysimm.system.System.center

Centers particles center of geometry or simulation box at given coordinate. A vector is defined based on the current coordinate for the center of either the particles or the simulation box and the "at" parameter. This shift vector is applied to the entity defined by the "what" parameter. Optionally, both the particles and the box can be shifted by the same vector.

Parameters

- **`what`** – what is being centered: "particles" or "box"
- **`at`** – new coordinate for center of particles or box
- **`move_both`** – if True, determines vector for shift defined by "what" and "at" parameters, and applies shift to both particles and box. If false, only shift what is defined by "what" parameter.

Returns None

`center_system`(*self*)
pysimm.system.System.center_system

DEPRECATED: Use *System*.center('box', [0, 0, 0], True) instead

Parameters `None` –

Returns None

`center_at_origin`(*self*)
pysimm.system.System.center_at_origin

DEPRECATED: Use *System*.center('particles', [0, 0, 0], True) instead

Parameters `None` –

Returns None

`set_mass`(*self*)
pysimm.system.System.set_mass

Set total mass of particles in *System*

Parameters `None` –

Returns None

`set_volume`(*self*)
pysimm.system.System.set_volume

Set volume of *System* based on Dimension

Parameters `None` –

Returns None

`set_density`(*self*)
pysimm.system.System.set_density

Calculate density of *System* from mass and volume

Parameters `None` –

Returns None

`set_velocity`(*self*)
pysimm.system.System.set_velocity

Calculate total velocity of particles in *System*

Parameters `None` –

Returns None

`zero_velocity`(*self*)
pysimm.system.System.zero_velocity

Enforce zero shift velocity in *System*

Parameters `None` –

Returns None

`set_box`(*self*, *padding=0.0*, *center=True*)
pysimm.system.System.set_box

Update *System*.dim with user defined padding. Used to construct a simulation box if it doesn't exist, or adjust the size of the simulation box following system modifications.

Parameters

- `padding` – add padding to all sides of box (Angstrom)
- `center` – if True, place center of box at origin default=True

Returns None

`set_mm_dist`(*self*, *molecules=None*)
pysimm.system.System.set_mm_dist

Calculate molecular mass distribution (mainly for polymer systems). Sets *System*.mw, *System*.mn, and *System*.disperisty

Parameters `molecules` – *ItemContainer* of molecules to calculate distributions defaul='all'

Returns None

`set_frac_free_volume`(*self*, *v_void=None*)
pysimm.system.System.set_frac_free_volume

Calculates fractional free volume from void volume and bulk density

Parameters `v_void` – void volume if not defined in *System*.void_volume default=None

Returns None

`visualize`(*self*, *vis_exec='vmd'*, ***kwargs*)

pysimm.system.System.visualize

Visualize system in third party software with given executable. Software must accept pdb or xyz as first command line argument.

Parameters

- **vis_exec** – executable to launch visualization software default='vmd'
- **unwrap** (*optional*) – if True, unwrap *System* first default=None
- **format** (*optional*) – set format default='xyz'

Returns None

`viz`(*self*, ***kwargs*)

`class pysimm.system.Molecule`(***kwargs*)

Bases: *System*

pysimm.system.Molecule

Very similar to *System*, but requires less information

`pysimm.system.read_yaml`(*file_*, ***kwargs*)

pysimm.system.read_yaml

Interprets yaml file and creates *System* object

Parameters **file** – yaml file name

Returns *System* object

`pysimm.system.read_xyz`(*file_*, ***kwargs*)

pysimm.system.read_xyz

Interprets xyz file and creates *System* object

Parameters

- **file** – xyz file name
- **quiet** (*optional*) – if False, print status

Returns *System* object

`pysimm.system.read_chemdoodle_json`(*file_*, ***kwargs*)

pysimm.system.read_chemdoodle_json

Interprets ChemDoodle JSON (Java Script Object Notation) file and creates *System* object

Parameters

- **file** – json file name
- **quiet** (*optional*) – if False, print status

Returns *System* object

`pysimm.system.read_lammps`(*data_file*, ***kwargs*)

pysimm.system.read_lammps

Interprets LAMMPS data file and creates *System* object

Parameters

- **data_file** – LAMMPS data file name
- **quiet** (*optional*) – if False, print status
- **atom_style** (*optional*) – option to let user override (understands charge, molecular, full)
- **pair_style** (*optional*) – option to let user override
- **bond_style** (*optional*) – option to let user override
- **angle_style** (*optional*) – option to let user override
- **dihedral_style** (*optional*) – option to let user override
- **improper_style** (*optional*) – option to let user override
- **set_types** (*optional*) – if True, objectify default=True
- **name** (*optional*) – provide name for system

Returns *System* object

`pysimm.system.`**`read_pubchem_smiles`**(*smiles*, *quiet=False*, *type_with=None*)

pysimm.system.read_pubchem_smiles

Interface with pubchem restful API to create molecular system from SMILES format

Parameters

- **smiles** – smiles formatted string of molecule
- **type_with** – *Forcefield* object to type with default=None

Returns *System* object

`pysimm.system.`**`read_pubchem_cid`**(*cid*, *type_with=None*)

pysimm.system.read_pubchem_smiles

Interface with pubchem restful API to create molecular system from SMILES format

Parameters

- **smiles** – smiles formatted string of molecule
- **type_with** – *Forcefield* object to type with default=None

Returns *System* object

`pysimm.system.`**`read_cml`**(*cml_file*, ***kwargs*)

pysimm.system.read_cml

Interprets cml file and creates *System* object

Parameters

- **cml_file** – cml file name
- **linkers** (*optional*) – if True, use spinMultiplicity to determine linker default=None

Returns *System* object

`pysimm.system.`**`read_mol`**(*mol_file*, *type_with=None*, *version='V2000'*)

pysimm.system.read_mol

Interprets mol file and creates *System* object

Parameters

- **mol_file** – mol file name

- **f** (*optional*) – *Forcefield* object to get data from
- **version** – version of mol file to expect default='V2000'

Returns *System* object

`pysimm.system.`**`read_mol2`**(*mol2_file*, *type_with=None*)
pysimm.system.read_mol2

Interprets .mol2 file and creates *System* object

Parameters

- **mol_file2** – a full name (including path) of a Tripos Mol2 text file
- **type_with** (*optional*) – *Forcefield* object to use for attempt to assighn
- **system** (*forcefield parameters to the*) –

Returns *System* object

`pysimm.system.`**`read_prepc`**(*prec_file*)
pysimm.system.read_prepc

Interprets prepc file and creates *System* object

Parameters **prepc_file** – ac file name

Returns *System* object

`pysimm.system.`**`read_ac`**(*ac_file*)
pysimm.system.read_ac

Interprets ac file and creates *System* object

Parameters **ac_file** – ac file name

Returns *System* object

`pysimm.system.`**`read_pdb`**(*pdb_file*, *str_file=None*, ***kwargs*)
pysimm.system.read_pdb

Interprets pdb file and creates *System* object

Parameters **pdb_file** – pdb file name

Keyword Arguments

- **str_file** – (str) optional CHARMM topology (stream) file which can be used as source of charges and description
- **topology** (*of bonded*) –
- **use_ptypes** – (bool) flag to either use the forcefield atom type names from the .str file or not

Returns *System* object

`pysimm.system.`**`compare`**(*s1*, *s2*)

`pysimm.system.`**`get_types`**(**arg*, ***kwargs*)
pysimm.system.get_types

Get unique type names from list of systems

Parameters **write** (*optional*) – if True, write types dictionary to filename

Returns (ptypes, btypes, atypes, dtypes, itypes) *** for use with update_types ***

pysimm.system.**distance_to_origin**(*p*)
pysimm.system.distance_to_origin

Calculates distance of particle to origin.

Parameters **p** – Particle object with x, y, and z attributes

Returns Distance of particle to origin

pysimm.system.**replicate**(*ref*, *nrep*, *s_=None*, *density=0.3*, *rand=True*, *print_insertions=True*)
pysimm.system.replicate

Replicates list of *System* objects into new (or exisintg) *System*. Can be random insertion.

Parameters

- **ref** – reference :class:`~pysimm.system.System`(s) (this can be a list)
- **nrep** – number of insertions to perform (can be list but must match length of ref)
- **s** – *System* into which insertions will be performed default=None
- **density** – density of new *System* default=0.3 (set to None to not change box)
- **rand** – if True, random insertion is performed
- **print_insertions** – if True, update screen with number of insertions

pysimm.utils

Module Contents

Classes

Container	pysimm.utils.Container
ItemContainer	pysimm.utils.ItemContainer
Item	

Functions

compare(query, item, query_wildcard=None, item_wildcard='X', order=False, improper_type=False)

exception pysimm.utils.**PysimmError**
Bases: `Exception`

Common base class for all non-exit exceptions.

class pysimm.utils.**Container**
Bases: `object`

pysimm.utils.Container

Abitrary container object that returns None if trying to access an attribute that does not exist

__getattr__(*self*, *name*)

class pysimm.utils.**ItemContainer**(*_dict=None*, ***kwargs*)

Bases: `collections.abc.Sequence`

pysimm.utils.ItemContainer

Container object intended to organize *`Item`* objects. Arbitrary attributes can be set using keyword arguments. Underlying data structure is a dictionary where the key is referred to as a tag, and the value should be an *`Item`* object. *`Item`*.tag should equal the key for the object in the dictionary.

`__len__`(*self*)

`__iter__`(*self*)

`__getitem__`(*self*, *slice_*)

add(*self*, *_item*)

get(*self*, **args*, ***kwargs*)

remove(*self*, *index*, *update=True*)

class pysimm.utils.**Item**(***kwargs*)

Bases: `object`

`__getattr__`(*self*, *name*)

copy(*self*)

set(*self*, ***kwargs*)

pysimm.utils.**compare**(*query*, *item*, *query_wildcard=None*, *item_wildcard='X'*, *order=False*, *improper_type=False*)

1.1.3 Package Contents

pysimm.**`__version__`** = **0.2.3**

pysimm.**error** = **True**

pysimm.**warning** = **True**

pysimm.**verbose** = **True**

pysimm.**debug** = **True**

pysimm.**error_print**

pysimm.**warning_print**

pysimm.**verbose_print**

pysimm.**debug_print**

exception pysimm.**PysimmError**

Bases: `Exception`

Common base class for all non-exit exceptions.

CHAPTER TWO

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

INDEX