CASSANDRA

Computational Atomistic Simulation Software At Notre Dame for Research Advances

User Manual 1.0

Written by:

Edward J. Maginn, Jindal K. Shah, Eliseo MarinRimoldi,
Sandip Khan, Neeraj Rai, Thomas Rosch, Andrew Paluch

© University of Notre Dame du Lac, 2014

Preface and Disclaimer

Cassandra is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, either version 3 of the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

This user manual is distributed along with the Cassandra software to aid in setting up various input files required for carrying out a Cassandra Monte Carlo simulation. Every effort is made to release the most updated and complete version of the manual when a new version of the software is released. To report any inconsistencies, errors or missing information, or to suggest improvements, send email to Edward Maginn (ed@nd.edu).

Acknowledgements

Support for this work was provided by a grant from the National Science Foundation entitled "SI2-SSE: Development of Cassandra, a General, Efficient and Parallel Monte Carlo Multiscale Modeling Software Platform for Materials Research", grant number ACI-1339785.

Ed Maginn would like to acknowledge financial support from Sandia National Laboratory's Computer Science Research Institute, which enabled him to take a research leave and lay the foundation for Cassandra in collaboration with Jindal Shah, who stayed behind at Notre Dame and helped keep the group going. The hospitality of Steve Plimpton and co-workers at Sandia is gratefully acknowledged.

Finally, we would also like to thank the Center for Research Computing at Notre Dame, which provided support, encouragement, and infrastructure to help bring Cassandra to life.

People who have contributed to Cassandra through algorithm development and / or writing code (to date) include:

- Ed Maginn
- Jindal Shah
- Tom Rosch
- Neeraj Rai
- Eliseo Marin
- Sandip Khan
- Andrew Paluch

Some legacy code was used in the creation of Cassandra, and the following former students are recognized for their work:

- David Eike
- Jim Larentzos
- Craig Powers

Contents

Chapter 1

Introduction

Cassandra is an open source Monte Carlo package capable of simulating any number of molecules composed of rings, chains, or both. It can be used to simulate compounds such as small organic molecules, oligomers, aqueous solutions and ionic liquids. It handles a standard "Class I"-type force field having fixed bond lengths, harmonic bond angles and improper angles, a CHARMM or OPLS-style dihedral potential, a Lennard-Jones 12-6 potential and fixed partial charges. It does *not* treat flexible bond lengths. Cassandra uses OpenMP parallelization and comes with a number of scripts, utilities and examples to help with simulation setup.

Cassandra is capable of simulating systems in the following ensembles:

- Canonical (NVT)
- Isothermal-Isobaric (NPT)
- Grand canonical (μVT)
- Constant volume Gibbs (NVT-Gibbs)
- Constant Pressure Gibbs (NPT- Gibbs)

1.1 Distribution

Cassandra is distributed as a tar file `cassandra.tar`. You can unpack the distribution by running the command

```
> tar -xvf cassandra.tar
```

Upon successful unpacking of the archive file, the Cassandra directory will have a number of subdirectories. Please refer to the README file in the main Cassandra directory for a detailed information on each of the subdirectories.

`Documentation` - Contains this user guide and a document showing how molecular connectivity files are

generated.

`Examples` - Contains example input files and short simulations of various systems in the above ensembles.

`MCF` - Molecular connectivity files for a number of molecules. These can be used as the basis for generating your own MCF files for molecules of interest.

`Scripts` - Useful scripts to set up simulation input files.

`Src` - Cassandra source code.

Chapter 2

Force Field

2.1 Bonds

Cassandra is designed assuming all bond lengths are fixed. If you wish to utilize a force field developed with flexible bond lengths, we recommend that you either use the nominal or "equilibrium" bond lengths of the force field as the fixed bond lengths specified for a Cassandra simulation or carry out an energy minimization of the molecule with a package that treats flexible bond lengths and utilize the bond lengths obtained from the minimization.

2.2 Angles

Cassandra supports two types of bond angles:

- 'fixed' : The angle declared as fixed is not perturbed during the course of the simulation.
- 'harmonic' : The bond angle energy is calculated as

$$E_\theta = K_\theta(\theta - \theta_0)^2 \tag{2.1}$$

  where the user must specify K_θ and θ_0. Note that a factor of $1/2$ is **not used** in the energy calculation of a bond angle. Make sure you know how the force constant is defined in any force field you use.

2.3 Dihedrals

Cassandra can handle four different types of dihedral angles:

- 'OPLS': The functional form of the dihedral potential is

$$E_\phi = a_0 + a_1\ (1 + \cos(\phi)) + a_2\ (1 - \cos(2\phi)) + a_3\ (1 + \cos(3\phi)) \tag{2.2}$$

  where a_0, a_1, a_2 and a_3 are specified by the user.

- 'CHARMM': The functional form of the potential is

$$E_\phi = a_0 * (1 + \cos(a_1 * \phi - \delta)) \tag{2.3}$$

  where a_0, a_1 and δ are specified by the user.

- 'harmonic': The dihedral potential is of the form:

$$E_\phi = K_\phi(\phi - \phi_0)^2 \tag{2.4}$$

  where K_ϕ and ϕ_0 are specified by the user.

- 'none' : There is no dihedral potential between the given atoms.

2.4 Impropers

Improper energy calculations can be carried out with the following two options:

- 'none': The improper energy is set to zero for the improper angle.

- 'harmonic': The following functional form is used to calculate the energy due to an improper angle

$$E_\psi = K_\psi \left(\psi - \psi_0\right)^2 \tag{2.5}$$

where K_ψ and ψ_0 are specified by the user.

2.5 Nonbonded

The nonbonded interactions between two atoms i and j are due to repulsion-dispersion interactions and electrostatic interactions (if any).

2.5.1 Repulsion-Dispersion Interactions

The repulsion-dispersion interactions can take one of the following forms:

- Lennard-Jones 12-6 potential (LJ):

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] \tag{2.6}$$

  where ϵ_{ij} and σ_{ij} are the energy and size parameters set by the user. For unlike interactions, different combining rules can be used, as described elsewhere. Note that this option only evaluates the energy up to a specified cutoff distance. As described below, analytic tail corrections to the pressure and energy can be specified to account for the finite cutoff distance.

- Cut and shift potential:

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] - 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{cut}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{cut}}\right)^{6}\right] \tag{2.7}$$

  where ϵ_{ij} and σ_{ij} are the energy and size parameters set by the user and r_{cut} is the cutoff distance. This option forces the potential energy to be zero at the cutoff distance. For unlike interactions, different combining rules can be used, as described elsewhere.

- Cut and switch potential:

$$\mathcal{V}(r_{ij}) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r_{ij}}\right)^{12} - \left(\frac{\sigma_{ij}}{r_{ij}}\right)^{6}\right] f \tag{2.8}$$

  The factor f takes the following values:

$$f = \begin{cases} 1.0 & r_{ij} \leq r_{on} \\ \frac{(r_{off}^2 - r_{ij}^2)(r_{off}^2 - r_{on}^2 + 2r_{ij}^2)}{\left(r_{off}^2 - r_{on}^2\right)^3} & r_{on} < r_{ij} < r_{off} \\ 0.0 & r_{ij} \geq r_{off} \end{cases} \tag{2.9}$$

  where ϵ_{ij} and σ_{ij} are the energy and size parameters set by the user. This option smoothly forces the potential to go to zero at a distance r_{off}, and begins altering the potential at a distance of r_{on}. Both of these parameters must be specified by the user. For unlike interactions, different combining rules can be used, as described elsewhere.

- Tail corrections: If the Lennard-Jones potential is used, standard Lennard-Jones tail corrections are used to approximate the long range dispersion interactions

2.5.2 Electrostatics

Electrostatic interactions are given by Coulomb's law

$$\mathcal{V}_{elec}(r_{ij}) = \frac{1}{4\pi\epsilon_0}\frac{q_i q_j}{r_{ij}}. \tag{2.10}$$

where q_i and q_j are partial charges specified by the user and placed on atomic positions given by r_i and r_j. In a simulation, the electrostatic interactions are calculated using either an Ewald summation or a direct summation using the minimum image convention.

Note that the total energy that is printed out in the property file is extensive. Consequently, to obtain intensive energies, the printed energies must divided by the total number of molecules in the system.

2.6 Cassandra Units

The following table provides units used in Cassandra:

Table 2.1: Cassandra units for input variables

Bond length	l	Å
Angles		
Nominal bond angle	θ_0	degrees
Bond angle force constant	K_θ	K/rad^2
Dihedral angle		
OPLS	a_0, a_1, a_2, a_3	kJ/mol
CHARMM	a_0	kJ/mol
	a_1	dimensionless
	δ	degrees
harmonic	K_ϕ	K/rad^2
	ϕ_0	degrees
Improper angle		
Force constant	K_ψ	K/rad^2
	ψ_0	degrees
Nonbonded		
Energy parameter	ϵ/k_B	K
Collision diameter	σ	Å
Charge	q	e
Simulation Parameters		
Simulation box length		Å
Volume		Å^3
Distances		Å
Rotational width		degrees
Temperature		K
Pressure		bar
Chemical potential		kJ/mol
Energy		kJ/mol
Fugacity		bar

Chapter 3

Cassandra Basics

3.1 Flow Diagram

A flow diagram that overviews the setup for a Cassandra simulation is displayed on figure 3.1. This diagram employs two automation scripts located in the `/Scripts/` directory: `mcfgen.py` and `library_setup.py`. These scripts are particularly useful when simulating large molecules. For details about each step, please refer to the document `MCF_tutorial.pdf`, to the README files located in the subdirectories inside the directory `/Scripts/`, and to this document.

3.2 Cassandra Simulation Setup

Once a system is identified, setting up a Cassandra simulation from scratch requires preparation of the following files.

- A molecular connectivity file (MCF) (*.mcf) containing the molecular connectivity information on bonds, angles, dihedrals, impropers and whether the molecule is composed of fragments. For information on the MCF file, please refer to section 4.2.
- An input file (*.inp) (See section 4.1)
- If the molecule is composed of fragments, then a fragment library file for each of the fragments is required. For instructions on how to generate these files, please refer to the document `MCF_tutorial.pdf`.

MCF files for united-atom models of methane, isobutane, dimethylhexane, cyclohexane and diethylether are provided in the `MCF` directory. Input files for NVT, NPT, GCMC and GEMC ensembles are located in the `Examples` directory which also contains fragment library files for a number of molecules simulated in these ensembles.

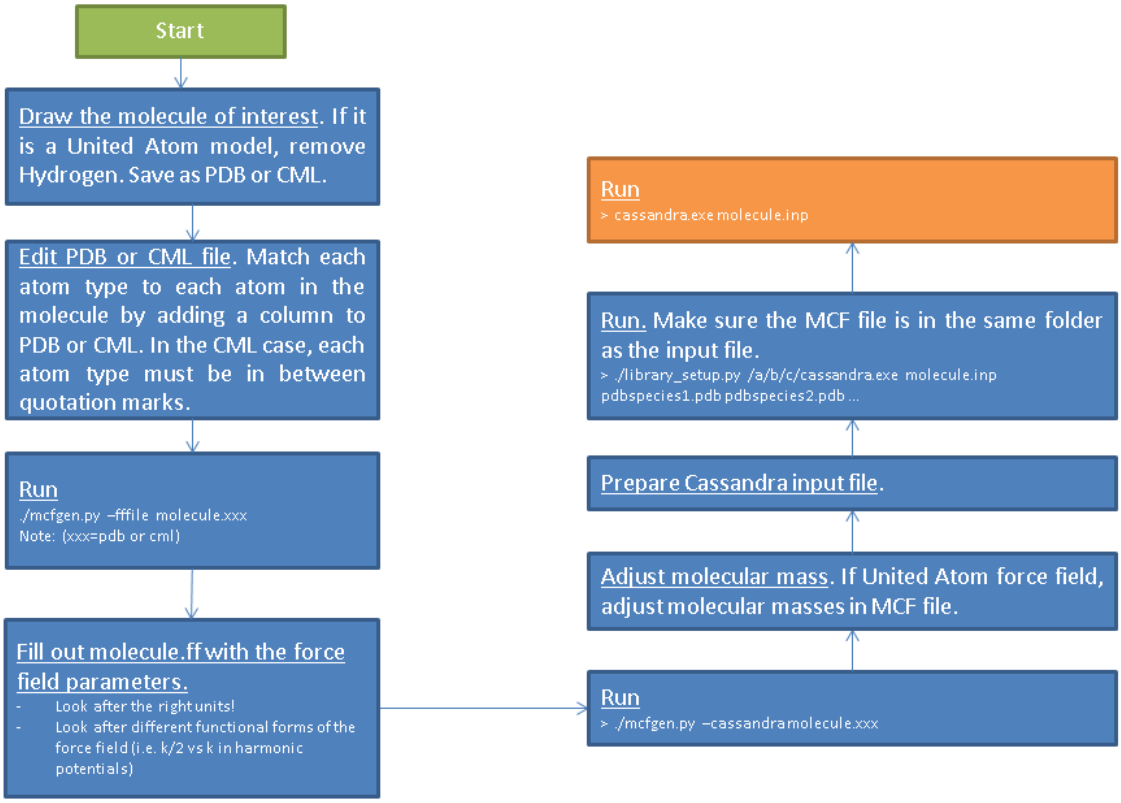

Figure 3.1: Flow diagram representing a typical setup of a Cassandra simulation

3.3 Cassandra File Preparation

3.3.1 MCF File

One MCF file is required for each unique species in a simulation. A species is defined as a collection of atoms associated with each other through bonds. Thus a molecule is a species as is an ion. If you wanted to simulate sodium sulfate, you would need separate MCF files for the sodium ion and the sulfate ion. MCF files can be built "by hand" or by using the scripts provided with the code, as described in the `Documentation/MCF_tutorial.pdf`. Instructions for generating an MCF file can also be found in the `Scripts/MCF_Generation/README` file. We will collect MCF files submitted to us by users and will post them on the Cassandra website http://cassandra.nd.edu. If you have an MCF file you would like us to post, send it to ed@nd.edu.

3.3.2 Input File

An input file is required for a Cassandra simulation. The input file specifies conditions for the simulation and various keywords required for the simulation in a given ensemble. Please refer to Chapter 4.1 for further details.

3.3.3 Fragment Library Generation

Cassandra makes use of reservoir sampling schemes to sample correctly and efficiently various coupled intramolecular degrees of freedom associated with branch points and rings. Details may be found in our publication [1]. The idea is to decompose the entire molecule into fragments that are either branch points or ring groups, each coupled to other fragments via a single dihedral angle. Thus, the total number of fragments of a molecule is the sum of branch points and ring groups in the molecule. The neighboring fragments are connected by two common atoms present in each of the fragments. Note that the ring group contains all the ring atoms and those directly bonded to the ring atoms. For each fragment identified, Cassandra runs a pre–simulation in the gas phase to sample the intramolecular degrees of freedom. A library of a large number of these conformations are stored for use in an actual simulation.

The gas phase library generation has been automated with the script `library_setup.py` located in the `Scripts/Frag_Library_Setup` directory. Use the following command for generating the fragment library.

```
> python path_to_Scripts/Frag_Library_Setup/library_setup.py cassandra_executable
input_filename mol1.pdb (mol1.cml) mol2.pdb (mol2.cml) ...
```

where `input_filename` is the name of the input file for the actual simulation and `mol1.pdb mol2.pdb ...` or `mol1.cml mol2.cml ...` correspond to the names of the pdb (or cml) files used to generated the MCF files. Make sure that if a file does not exist in the current working directory, its path relative to the current working directory is specified.

3.4 Running a Simulation

To launch a Cassandra simulation, run the following command:

```
> cassandra_executable input_filename
```

The executable will read `input_filename` and execute the instructions. Make sure that the required files (MCF, fragment library files) are located in the directories as given in the input file.

3.5 Restarting a Simulation

Restarting a simulation requires either a checkpoint file (*.chk produced by Cassandra) or a configuration file obtained from xyz files generated from a previous simulation. For the set up of these simulations, a script in `Scripts/Read_Old` is provided. Detailed instructions are contained in the README file in this directory.

3.6 Cassandra Output Files

Cassandra generates several output files which can be used for later analysis. All have as a prefix the Run_Name specified in the input file. See Chapter 4.1 for details. The type of output is specified by the file name suffix. The following are generated:

- **Log file** (*.log): Contains basic information on what the run is, timing information and reports the various parameters specified by the user. A complete copy of the input file is reproduced. Other important information includes the move acceptance rates. You can use the log file to keep track of what conditions were simulated.
- **Coordinate file** (*.xyz): For each box in the system, a set of xyz coordinates are written out with a frequency specified by the user (Ncoordfreq). The file has as a header the number of atoms in the box. Following this, the atomic coordinates of molecule 1 of species 1 are written, then the coordinates of molecule 2 of species 1 are written, etc. After all the coordinates of the molecules of species 1 are written, the coordinates of the molecules of species 2 are written, etc. You can use this file to do all your structural analysis and post processing.
- **Checkpoint file** (*.chk): A checkpoint file is written every Ncoordfreq steps. This can be used to restart a simulation from this point using all of the same information as the run that was used to generate the checkpoint file. To do this, you must use the checkpoint restart option (see Chapter 4.1). It will basically pick up where the simulation left off, using the same random number seed, maximum displacements, etc. This is useful in case your job crashes and you want to continue running a job. You can also use the checkpoint file to start a new simulation using the configuration of the checkpoint file as an initial configuration and the optimized maximum displacements. To do this, use the script read_old.py. You will need to set a new random number seed if you do this. See the documentation in Chapter 4.1 for more details.
- **H-matrix file** (*_H.box#): This file is written to every Ncoordfreq MC steps. The first line is the box volume in angstrom^3. The next three lines are the box coordinates in angstrom in an H-matrix form. Since Cassandra only supports cubic boxes at the moment, this is just a diagonal and symmetric matrix, but is included here for later versions that will enable non-orthogonal boxes. After this, a blank line is written. The next line is the box number, and the final line(s) is(are) the species ID and number of molecules for that species in this box. If there are three species, there will be three lines. This output is repeated every Ncoordfreq times. This file allows you to compute the density of the box during constant pressure simulations.
- **Property file** (*.instnt_prp#_box#): This file lists the instantaneous thermodynamic and state properties for each box. Note that you can have more than one property file (hence the# after 'prp') and more than one box (also why there is a # after 'box'). The user specifies which properties are to be written and in what order, and these are then reproduced in this file. The file is written to every Nthermofreq steps. A header is written to the first two lines to designate what each property is. You may use this file to compute thermodynamic averages.
- **Initial configuration** (*.init_config_box#): If you generate your initial configuration using the make_config command, this file will be created for each box. It contains the initial coordinates of all the species in the system. You can use this to check on whether the initial configuration is reasonable, or use it as an input to other codes. For example, the initial configuration will be generated using CBMC, so it may be a better starting configuration that if you used other methods.

Chapter 4

Files Required to Run Cassandra

4.1 Simulation Input File

This is a required file that is given as an argument to the Cassandra executable. You must generate this by hand, but you can use the input files in the Examples directory as a guide. The input file contains a number of keywords that define simulation parameters and thermodynamic state point for the simulation. A keyword is identified by a '#' while comments follow a '!'. Note that the order of the keywords in the input file is immaterial, but the format of the arguments of the keyword are important and are explained below, along with a complete listing of all keywords.

4.1.1 Run Name

 # Run_Name - Simulation run name
*Character*120*

The run name is specified on the next line following the keyword. This name is used as a prefix for all the files produced by the simulation. For example,

```
# Run_Name
dee
```

Cassandra will then use `dee` as prefix for all output files created.

4.1.2 Simulation Type

 # Sim_Type
*Character*120*

Sets the ensemble (and thus the suite of moves) of a Cassandra simulation. Currently the following ensembles are supported:

- NVT_MC (Canonical ensemble)
- NVT_MIN (Canonical ensemble in which minimization is carried out)
- NPT_MC (Isothermal-isobaric ensemble)
- GCMC (Grand canonical ensemble)
- GEMC (Gibbs ensemble)
- NVT_MC_Fragment (Canonical ensemble simulation of a non-ring fragment)
- NVT_MC_Ring_Fragment (Canonical ensemble simulation of a ring fragment)

Note that NVT_MC_Fragment and NVT_MC_Ring_Fragment are used only for the fragment generation and are not used in the normal simulation. For example,

```
# Sim_Type
NPT_MC
```

will run an NPT MC simulation.

4.1.3 Number of species

 # Nbr_Species
Integer

Total number of species in the simulation. For ionic systems, each ion is counted as a separate species. For example, for a mixture of two species, use the following:

```
# Nbr_Species
2
```

4.1.4 VDW Style

 # VDW_Style
Character(i,1) Character(i,2) Real(i,3) Real(i,4)/Logical(i,4)

This keyword specifies the functional form of repulsion dispersion interactions to be used and if tail corrections are added for the box *i*. One line is required for each box. The options for *Character(i,1)* are "LJ" for a Lennard-Jones 12-6 potential or "None" if you wish to turn off all repulsion-dispersion interactions. *Character(i,2)* and *Real(i,3)* are specified only if *Character(i,1)* is set to "LJ". *Character(i,2)* describes the truncation scheme used for the Lennard-Jones potential. Options are "cut", "cut_tail", "cut_switch" and "cut_shift". Refer to Chapter **??** for the functional forms. The other parameters *Real(i,3)* and *Real(i,4)/Logical(i,4)* depend on the selection of *Character(i,2)* as described below:

`cut`: This option cuts the potential off at the distance specified by (*Real(i,3)*). The fourth parameter is omitted.

For example, to simulate one box with a 14 Å cutoff specify the following:

```
# VDW_Style
LJ cut 14.0
```

Similarly, for a two box simulations such as used in the Gibbs ensemble where both boxes have a 14 Å cutoff, use the following:

```
# VDW_Style
LJ cut 14.0
LJ cut 14.0
```

`cut_tail`: This options cuts the potential off at a distance corresponding to (*Real(i,3)*) and applies analytic tail corrections to the energy and pressure. An optional fourth argument (*Logical(i,4)*) can be set to 'TRUE' or 'true', in which case *Real(i,3)* is ignored and the cutoff distance is always set to half of the simulation box length. The cutoff will change during the course of the simulation when attempting volume moves. This option is provided to enable reproduction of literature simulations that use a cut off distance of half the simulation box length, but its use is highly discouraged.

For example, to simulate one box with a 14 Å cutoff using tail corrections, specify the following:

```
# VDW_Style
LJ cut_tail 14.0
```

For a two box simulation where the first box has a 14 Å cutoff and the second one has a 20 Å cutoff, use the following:

```
# VDW_Style
LJ cut_tail 14.0
LJ cut_tail 20.0
```

`cut_switch`: This option cuts the potential off and smoothly brings the potential to zero using a spline. The potential is cutoff and the spline turned on at a distance specified by *Real(i,3)* (r_{on} in Eq 2.8) and the potential goes to zero at a distance specified by *Real(i,4)* (r_{off} in Eq 2.8).

For example, a one box simulation using the cut_switch option could be specified as follows:

```
# VDW_Style
LJ cut_switch 12.0 14.0
```

In this case, the Lennard-Jones potential would end at 12.0 Å and be smooothly taken to zero at 14.0 Å. Obviously, $r_{on} < r_{off}$ or *Real(i,3) < Real(i,4)*.

`cut_shift`: This option cuts the potential off at a distance specified by *Real(i,3)* and shifts the entire potential so that at this distance the potential is zero. The fourth parameter *Real(i,4)/Logical(i,4)* is omit-

ted. The functional form of this potential is given in eq 2.7.

To perform a two box simulation with a cut_shift option in which both boxes have a 10.5 Å cutoff, use the following:

```
# VDW_Style
LJ cut_shift 10.5
LJ cut_shift 10.5
```

Note: For all options, cutoff distances must be less than or equal to the shortest edge length of a simulation box.

4.1.5 Charge Style

 # Charge_Style
Character(i,1) Character(i,2) Real(i,3) Real(i,4)

Cassandra allows the use of fixed partial charges on atomic centers using a Coulomb potential of the form given in Eq 2.10. To specify this option for box *i*, set *Character(i,1)* to "coul". For this option, *Character(i,2)* can be set to either "Ewald" if you want to use an Ewald sum to compute Coulombic interactions or it can be set to "cut", in which case the Coulombic interactions will be cut off and the long range interactions ignored. For the Ewald option, *Real(i,3)* is the real space cutoff distance and *Real(i,4)* specifies the accuracy of the Ewald summation. A reasonable value for the accuracy is 10^{-5}. Note that the number of reciprocal vectors for the Ewald summation is determined in the code based on the accuracy parameter. For more details, see the paper by Fincham [2].

For example,

```
# Charge_Style
coul Ewald 12.0 1E-5
```

will use the Ewald sum for a single box. The real space cutoff will be 12 Å and the accuracy will be 10^{-5}. If you have two boxes, like in a Gibbs ensemble calculation, then you could use the following:

```
# Charge_Style
coul Ewald 12.0 1E-5
coul Ewald 30.0 1E-5
```

This will use an Ewald sum for both boxes. In the first box, the real space cutoff will be 12 Å while in the second box a larger cutoff of 30 Å will be used. **Note: When performing Gibbs ensemble simulations of vapor-liquid equilibria, the vapor box is often much larger than the liquid box. In this case, you will want to use a longer real space cutoff for the larger vapor box to avoid using too many reciprocal space vectors.** Also note that the real space cutoffs must always be less than or equal to half of the shortest edge length of a simulation box.

If you do not wish to use a Coulomb potential (for example, your model has no partial charges), you

still must specify # Charge_Style. In this case, set *Character(i,1)* to "NONE". If "NONE" is selected for *Character(i,1)* then *Character(i,2)*, *Real(i,3)* and *Real(i,4)* are omitted.

For example,

```
# Charge_Style
NONE
```

should be used if you have no partial charges and are simulating a single box. A two box simulation with no partial charges would be specified as

```
# Charge_Style
NONE
NONE
```

Note : If the cutoff in VDW_Style is set to half of the simulation box length, any cutoff distance specified in the Charge_Style section will default to the half of the simulation box length. In the case of Ewald summation, however, the accuracy will be the same as *Real(i,4)*.

4.1.6 Intramolecular Scaling

 # Intra_Scaling
Real(i,1) Real(i,2) Real(i,3) Real(i,4)
Real(i,5) Real(i,6) Real(i,7) Real(i,8)

This keyword sets the intramolecular scaling for 1-2, 1-3, 1-4 and 1-N interactions within a given species. 1-2 means interactions between a given atom 1 and another atom 2 directly bonded to it, 1-3 means interactions between atom 1 and other atoms 3 separated from atom 1 by exactly two bonds, etc. The first line corresponds to the VDW scaling: *Real(i,1) Real(i,2) Real(i,3) Real(i,4)* apply to 1-2, 1-3, 1-4 and 1-N interactions, where N corresponds to all atoms separated from atom 1 by more than three bonds. The second line corresponds to the Coulomb scaling: *Real(i,5) Real(i,6) Real(i,7) Real(i,8)* apply to 1-2, 1-3, 1-4 and 1-N interactions. These lines are repeated for each species in the simulation. Note that intramolecular scaling applies to all the boxes in the simulation.

For example,

```
# Intra_Scaling
0.0 0.0 0.5 1.0
0.0 0.0 0.5 1.0
```

would turn off 1-2 and 1-3 interactions, would scale the VDW and Coulombic interactions for 1-4 atoms by 50% and would use full interactions for all other atom pairs in the species.

If you had two species in the simulation and wanted the same intramolecular scaling as above, you would specify

```
# Intra_Scaling
0.0 0.0 0.5 1.0
0.0 0.0 0.5 1.0
0.0 0.0 0.5 1.0
0.0 0.0 0.5 1.0
```

In the absence of the # Intra_Scaling keyword, default values of 0.0, 0.0, 0.5 and 1.0 will be used for 1-2, 1-3, 1-4 and 1-N for VDW and Coulomb interactions for all the species in the simulation. If # Charge_Scaling is set to "NONE", you must still specify the Intra_Scaling for Coulombic interactions, but no interactions will be computed.

4.1.7 Mixing Rule

 # Mixing_Rule
Character

Sets the method by which Lennard-Jones interactions between unlike atoms are calculated. Acceptable options are "LB" for Lorentz-Berthelot and "geometric" for geometric. If this keyword is missing, "LB" is used as default.

4.1.8 Starting Seed

 # Seed_Info
Integer(1) Integer(2)

Inputs for the starting random number seeds for the simulation. Note that Cassandra uses a random number generator proposed by L'Ecuyer [3], which takes five seeds to calculate a random number, out of which three are defined internally while two *Integer(1)* and *Integer(2)* are supplied by the user. **When a 'checkpoint' file is used to restart a simulation (see # Start_Type below), the user supplied seeds will be overwritten by those present in the checkpoint file. If # Start_Type is set to 'read_old', then the seeds specified in the input file are used**.

As an example,

```
# Seed_Info
1244432 8263662
```

is an acceptable way of specifying the seeds. Note that two independent simulations can be run using the same input information if different seeds are used. If two simulations having exactly the same input information and the same seeds are run, the results will be identical.

4.1.9 Minimum Cutoff

 # Rcutoff_Low
Real(1)

Sets the minimum distance *Real(1)* in Å such that any MC move bringing two sites closer than this distance will be immediately rejected. It avoids numerical problems associated with random moves that happen to place atoms very close to one another such that they will have unphysically strong repulsion or attraction. This distance must be less than the intramolecular distance of all atoms in a species which are not bonded to one another. For models that use dummy sites without explicitly defining bonds between dummy and atomic sites of the molecules (for example, the TIP4P water model), it is important that the minimum distance is set to be less than the shortest distance between any two sites on the molecule. For most systems, 1 Å seems to work OK, but for models with dummy sites, a shorter value may be required.

4.1.10 Pair Energy Storage

 # Pair_Energy
Logical(1)

Cassandra can use a time saving feature in which the energies between molecules are stored and used during energy evaluations after a move, thereby saving a loop over all molecules. This requires more memory, but it can be faster. The default is to not use this feature. If you wish to use this, set *Logical(1)* to 'TRUE' or 'true'.

4.1.11 Molecule Files

 # Molecule_Files
Character(i,1) Integer(i,2)

This specifies the name of the molecular connectivity file (*.mcf) and the maximum total number of molecules of a given species specified by this MCF file. A separate line is required for each species present in the simulation. *Character(i,1)* is the name of the MCF file for species *i*. *Integer(i,2)* denotes the maximum number of molecules expected for the species.

For example

```
# Molecule_Files
butane.mcf 100
hexane.mcf 20
octane.mcf 5
```

specifies that there are three different species, and the MCF files state the names of the files where information on the three species can be found. Species 1 is butane, species 2 is hexane and species 3 is octane. There can be a maximum of 100 butane molecules, 20 hexane molecules and 5 octane molecules in the total system. The maximum number of molecules specified here will be used to allocate memory for each species, so do not use larger numbers than are needed.

4.1.12 Simulation Box

 # Box_Info
Integer(1)

Character(i,2)
Real(i,3)

This keyword sets parameters for the simulation boxes. *Integer(1)* specifies the total number of boxes in the simulation. For now, Gibbs ensemble simulations must have two boxes. *Character(i,2)* is the shape of the i^{th} simulation box. Right now, the only supported box shape in Cassandra is cubic, so use the keyword "CUBIC". *Real(i,3)* is the length of the box edges for a cubic box in Å. Information for additional boxes is provided in an analogous fashion and is separated from the previous box by a blank line. For a two box simulation, box information is given as:

 # Box_Info
```
2
CUBIC
30.0

CUBIC
60.0
```

This will construct two cubic boxes. The first will be 30 X 30 X 30 cubic Å and the second will be 60 X 60 X 60 cubic Å.

4.1.13 Temperature

 # Temperature_Info
Real(1) Real(2) ....

Sets the temperature in Kelvin *Real(1) Real(2) ...* for simulation boxes 1, 2, ...

4.1.14 Pressure

 # Pressure_Info
Real(1) Real(2) ...
Specifies the pressure in bar *Real(1) Real(2) ...* for simulation boxes 1, 2, ... Note that if the simulation type does not require an input pressure (for example, an NVT simulation), this command will be ignored.

4.1.15 Fugacity

 # Fugacity_Info
Real(1) Real(2) ...

Specifies the fugacities in bar *Real(1) Real(2) ...* of **insertable** species in the order in which they appear in the Molecule_Files section. The fugacity will be arbitrarily set to zero for species that cannot be swapped or exchanged with a reservoir. This option is used with grand canonical ensemble simulations. It will be ignored for all other simulation types.

4.1.16 Chemical Potential

 # Chemical_Potential_Info
Real(1) Real(2) ....

This keyword is only used for grand canonical simulations and may be used instead of fugacity. It sets the chemical potential of the insertable species in the order in which they appear in the `Molecule_Files` section. The chemical potentials will be set arbitrarily to zero for species that cannot be swapped or exchanged with a reservoir. Note that the de Broglie wavelength of the species in each box is automatically calculated and used in the acceptance rules. Units of chemical potential are kJ/mol. **Note: if you specify BOTH a fugacity AND a chemical potential, Cassandra will use whichever one appears first in the input file and will ignore the other keyword. We recommend that you do not list both keywords in any input file.**

4.1.17 Move Probabilities

 # Move_Probabilities This section specifies the probabilities associated with different types of MC moves to be performed during the course of simulation. Please ensure that the move probabilities add up to 1.0. An error will be generated if this is not the case. All the headers are optional but an error will be produced if a move is required (for example, volume fluctuations in an NPT simulation) and the move is not specified.

Translation

 # Prob_Translation
Real(1)
Real(i,j) *One line required for each box, and one value required for each species on each line.

Real(1) is the probability of performing a center of mass translation move. For each box i, the maximum displacement of species j is specified by *Real(i,j)*.

For example, if you have two species and two boxes, you would specify the translation probability as

```
# Prob_Translation
0.25
2.0 2.5
10.0 11.0
```

This will tell Cassandra to attempt center of mass translations 25% of the time. For box 1, the maximum displacement will be 2.0 Å for species 1 and 2.5 Å for species 2. For box 2, the maximum displacement for species 1 will be 10.0 Å and it will be 11.0 Å for species 2. Note that attempted moves will occur with equal probability for a give box, but attempts for a species are proportional to their mole fraction in the box.

Rotation

 # Prob_Rotation
Real(1)

Real(i,j) *One line required for each box, and one value required for each species on each line.

The probability of performing a rotation move is specified by *Real(1)* while *Real(i,j)* denotes the maximum roational width for species j in box i in degrees. If you are only simulating spherical molecules (such as Lennard-Jones particles), then do not use this keyword. If you are simulating a multi-species system where some of the species have rotational degrees of freedom and some species are spheres, then specify an appropriate value of *Real(i,j)* for the species having rotational degrees of freedom, and set *Real(i,j)* equal to zero for the spherical species. Linear molecules are a special case, where rotation is handled in Eulerian space. If you have a linear molecule such as carbon monoxide, specify any non-zero value for *Real(i,j)*. Cassandra will properly sample the rotational degrees of freedom but will not use the value set by *Real(i,j)*. Note that spherical molecules are not considered when choosing which species to perform a rotational move on.

For example, if you are simulating two species in two boxes and if the first species has rotational degrees of freedom while the second is spherical, you would specify the rotational probability as

```
# Prob_Rotation
0.25
30.0 0.0
180.0 0.0
```

This will tell Cassandra to perform rotational moves 25% of the time. For box 1, the maximum rotational width will be 30° for species 1 and 0.0° for species 2. For box 2, the maximum rotational width will be 180° for species 1 and 0.0° for species 2. Note that, since the maximum rotational width of species 2 is set to 0° in both boxes, no rotational moves will be attempted on species 2.

For a single box simulation with three species such that the first species has rotational degree of freedom, the second is a linear molecule and the third species is spherical, you would specify

```
# Prob_Rotation
0.25
30.0 10.0 0.0
```

This will tell Cassandra to attempt rotational move 25% of the time. The maximum rotational width for species 1 is 30° and that for species 2 is 10.0°. Since the species 2 is a linear molecule, its rotation will be attempted in Eulerian angles and Cassandra will not use this value. Since the rotational width is set equal to 0° for species 3, no rotational moves will be attempted for this species.

Regrowth

 # Prob_Regrowth
Real(1)
*Real(i,2)** One for each species

A regrowth move consists of deleting part of the molecule randomly and then regrowing the deleted part via a configurational bias algorithm. This can result in relatively substantial conformational changes for the molecule, but the cost of this move is higher than that of a simple translation or rotation. The probabil-

ity of attempting a regrowth move is specified by *Real(1)* while *Real(i,2)* specifies the relative probability of performing this move on species *i*. For monatomic species, *Real(i,2)* is set to zero. Note that the user needs to ensure that the relative probabilities add up to 1 otherwise Cassandra will display an error and quit.

For example, if you are simulating three species of which the first species is monatomic, you would specify the following:

```
# Prob_Regrowth
0.3
0.0 0.7 0.3
```

This will tell Cassandra to attempt regrowth move 30% of the time. The relative probabilities of performing regrowth moves on species 1, 2 and 3 are 0.0, 0.7 and 0.3 respectively.

Volume

 # Prob_Volume
Real(1)
Real(2,i) * One line required for each box except for GEMC-NVT

Sets the probability of volume displacement moves. This flag is required for NPT, GEMC-NPT and GEMC-NVT simulations. Do not specify for any other simulation type. *Real(1)* is the relative probability of attempting a box volume change. Note that volume changes are bold and expensive moves and should be attempted infrequently. This probability should normally not exceed 0.05 and values from 0.01-0.03 are typical. *Real(2,i)* specifies the maximum volume displacement in $Å^3$ for box *i*. If you are simulating a two box system, a value of *Real(2,i)* is required for each box on separate lines. Note that the exception to this is for a GEMC-NVT simulation, where there are two boxes but the volume moves are coupled. In this case, only a single value of *Real(2,i)* is specified. The default is to change the box volume by random amounts up to the maximum value specified by *Real(2,i)*. For example, if you are simulating a liquid with a single box in the NPT ensemble, you would specify the following:

```
# Prob_Volume
0.02
300
```

This will tell Cassandra to attempt volume moves 2% of the time. The box volume would be changed by random amounts ranging from -300 $Å^3$ to +300 $Å^3$. For a liquid box 20 Å per side, this would result in a maximum box edge length change of about 0.25 Å, which is a reasonable value. Larger volume changes should be used for vapor boxes. If you wish to perform a GEMC-NPT simulation, you might specify the following:

```
# Prob_Volume
0.02
300
5000
```

This will tell Cassandra to attempt volume moves 2% of the time. The first box volume (assumed here to be smaller and of higher density, such as would occur if it were the liquid box) would be changed by random amounts ranging from -300 Å^3 to +300 Å^3. The second box volume would be changed by random amounts ranging from -5000 Å^3 to +5000 Å^3. As with all move probabilities, you can experiment with making bolder or more conservative moves. Note that if the # Run_Type is set to 'Equilibration', Cassandra will attempt to optimize the boldness of moves to achieve about 50% acceptance rates.

Insertion and Deletion Moves

 # Prob_Insertion
Real(1)
Character(i,1)
Character(i,2)

This flag is set only for GCMC simulations. *Real(1)* sets the probability of attempting insetion moves. If there is more than one species, each is chosen for an insertion attempt with equal probability. *Character(i,1)* and *Character(i,2)* control the manner with which the insertions are carried out for each species *i*. Right now, Cassandra only uses one insertion method (a reservoir sampling approach). Later versions will have other options. So for now, the only option is to set *Character(i,1)* equal to 'insertion method'. If the species can be inserted or deleted, set *Character(i,2)* equal to 'reservoir'. If the species is 'non-volatile' and should not be inserted or deleted or should stay in its original box, then set *Character(i,2)* equal to 'none'. Then whichever box that species starts in, it will remain there for the whole simulation. You must repeat these flags for each species *i*. For example, if you are performing a GCMC simulation with two species that can be inserted, you might specify the following

```
# Prob_Insertion
0.1
insertion method
reservoir
insertion method
reservoir
```

This will tell Cassandra to attempt insertions 10% of the time and both species will be inserted using the reservoir insertion method.

```
# Prob_Insertion
0.1
insertion method
reservoir
insertion method
none
```

This will tell Cassandra to attempt insertions 10% of the time. Only species 1 will be inserted, while species 2 will not get inserted.

 # Prob_Deletion

Real (1)

Sets the relative probability of deletion during the course of a simulation. Each exchangeable species is randomly chosen and a deletion move is attempted on a randomly chosen molecule of this species. If a species has an insertion method 'none', no attempt is made to delete it. You must specify the same deletion probability as the insertion probability to satisfy microscopic reversibility. If you fail to do this, Cassandra will give an error and quit.

 # Prob_Swap
Real(1)
Character(i,1)
Character(i,2)

This keyword is set only for a GEMC simulation to enable transfer of species between two boxes. *Real(1)* sets the relative probability of attempting transfer of a molecule from one box to the other. During the swap, the donor and receiving boxes are chosen randomly. The species chosen for transfer is selected according to its overall mole fraction which is calculated only for the species that can be exchanged between boxes. Thus, species that are "non-volatile" are not included while computing the mole fractions. A molecule is then chosen randomly for transfer.

Similar to the `# Prob_Insertion` section, *Character(i,1)* and *Character(i,2)* describe the manner in which the swap is carried out for each species i. At present, the only option is to set *Character(i,1)* equal to 'insertion method'. If the species can be swapped, set *Character(i,2)* equal to 'reservoir'. If the species is not to be transferred between boxes, then set *Character(i,2)* to 'none'. Then whichever box that species starts in, it will remain in that box for the whole simulation. These flags are to be repeated for each species i. For example, while performing a GEMC simulation for three species the first two of which are exchanged while the third is not, you might specify the following:

```
# Prob_Swap
0.1
insertion method
reservoir
insertion method
reservoir
insertion method
none
```

This will tell Cassandra to attempt swap moves 10% of the time. Attempts will be made to transfer species 1 and 2 between available boxes while molecules of species 3 will remain in the boxes they are present in at the start of the simulation.

Flip Move

 # Prob_Ring
Real(1) Real(2)

This keyword is used when flip moves are to be attempted to sample bond angles and dihedral angles in a ring fragment. For more details on this move, see our publication [1]. The relative probability of attempting a flip move is specified by *Real(1)* while the maximum angular displacement in degrees for the move is given by *Real(2)*. For example, if the flip is to be attempted 30% of the time and the maximum angular displacement for the move is 20° specify the following:

```
# Prob_Ring
0.30 20.0
```

 # Done_Probability_Info
This is a required keyword that marks the end of the section for specifying move probabilities. It must occur after # Move_Probability and all the move probabilities must be specified between these two keywords. Once Cassandra reads # Done_Probability_info, it checks to make sure the probabilities sum to unity. If not, an error will be given.

4.1.18 Start Type

 # Start_Type
Character(1)

This keyword specifies whether Cassandra generates an initial configuration or uses a previously generated configuration to start a simulation. *Character(1)* takes one of the three options: 'make_config', 'checkpoint' or 'read_old' and it determines what configuration is used to start a simulation.

When 'make_config' is used as the start type, Cassandra will generate an initial configuration. With this option, additional information is required on the number of molecules of each species in every box and is specified as follows:

make_config
Integer(j,k) * One line for each species and one entry on each line for each box

where *Integer(j,k)* represents the number of molecules of species j in box k. Thus, for example, to generate an initial configuration for two species in two boxes such that the numbers of molecules of species 1 in box 1 and 2 are 100 and 50 respectively and those for species 2 are 75 and 25 respectively, the input file must contain the following:

```
# Start_Type
make_config
100 50
75 25
```

During the course of a simulation, Cassandra periodically generates a checkpoint file (*.chk) containing information about the total number of translation, rotation and volume moves along with the random number seeds and the coordinates of all the molecules and their box number at the time the file is written. Cassandra provides the capability of restarting from this state point in the event that a simulation crashes or running a production simulation from an equilibrated configuration. For this purpose, in addition to the

'checkpoint' keyword, additional information in the form of the name of the checkpoint file *Character(2)* is required in the following format:

checkpoint
Character(2)

For example, to continue simulations from a checkpoint file 'methane_vle_T148.chk', you might specify:

```
# Start_Type
checkpoint
methane_vle_T148.chk
```

Cassandra also provides a 'read_old' option to make use of just the coordinates of molecules to start a simulation. For example, a configuration generated at a lower temperature may be used to jump start a simulation at a higher temperature. When the 'read_old' option is used, additional information in the form of the file names *Character(k,3)* is required as shown below:

read_old
Character(k,3) * One line for each box

For example, to start a GEMC simulation simulation using the configurations of the two boxes, you might specify:

```
# Start_Type
read_old
box1.readold
box2.readold
```

This will tell Cassandra to use the configurations of the two boxes stored in `box1.readold` and `box2.readold` to start a simulation. Note that configurations of the boxes can be easily extracted from the checkpoint file using the utility `read_old.py` provided in `Scripts/Read_Old`.

4.1.19 Run Type

 # Run_Type
Character(1) Integer(1) Integer(2)

This keyword is used to specify whether a given simulation is an equilibration or a production run. For an equilibration run, the maximum translational, rotational and volume widths (for an NPT or a GEMC simulation) are adjusted to achieve 50% acceptance rates. During a production run, the maximum displacement width for different moves are held constant.

Depending on the type of the simulation, *Character(1)* can be set to either "Equilibration" or "Production". For an `Equilibration` run, *Integer(1)* denotes the number of MC steps performed for a given thermal move before the corresponding maximum displacement width is updated. *Integer(2)* is the number of MC volume moves after which the volume displacement width is updated. This number is optional if no volume moves

are performed during a simulation (for example in an NVT or a GCMC simulation). When the run type is set to `Production`, the MC moves refer to the frequency at which the acceptance ratios for various moves will be computed and output to the log file. These acceptance rates should be checked to make sure proper sampling is achieved.

For an NPT equilibration run in which the widths of the thermal move are to be updated after 100 MC moves and maximum volume displacements after 10 volume moves, specify the following:

```
# Run_Type
Equilibration 100 10
```

For an NVT production run in which the acceptance ratios of various thermal moves are printed to the log file after every 250 MC steps of a given thermal move, use the following:

```
# Run_Type
Production 250
```

4.1.20 Frequency

 # Frequency_Info
freq_type Character(1)
Nthermofreq or *thermofreq Integer(2)*
Ncoordfreq or *coordfreq Integer(3)*
MCsteps or *Stop Integer(4)*
 # Done_Frequency_Info

This section specifies the frequency at which thermodynamic properties and coordinates are output to a file. *Character(1)* determines the method by which the simulation is terminated and data is output. If *Character(1)* is to 'Timed', then the simulation stops after the specified time in minutes. The format for this option is given below:

freq_type Timed
thermofreq *Integer(2)*
coordfreq *Integer(3)*
Stop *Integer(4)*

With this option, thermodynamic quantities are output every *Integer(2)* minutes, coordinates are written to the disk every *Integer(3)* minutes and the total simulation time is specified in minutes by *Integer(4)*. For example, to run a simulation for 60 minutes such that thermodynamic quantities are written every minute and the coordinates are output every 10 minutes, use the following:

```
# Frequency_Info
freq_type Timed
thermofreq 1
coordfreq 10
Stop 60
```

```
# Done_Frequency_Info
```

Note that similar to `# Move_Probabilities` section, the end of the frequency section always includes the `# Done_Frequency_Info` line.

Simulations can also be run for a given number of MC steps. To enable this feature, *Character(1)* is set to 'none'. Additional information is required and is given in the following format:

freq_type none
Nthermofreq *Integer(2)*
Ncoordfreq *Integer(3)*
MCsteps *Integer(4)*

With this option, thermodynamic quantities are output every *Integer(2)* MC steps, coordinates are written at a frequency of *Integer(3)* MC steps and the simulation terminates after *Integer(4)* steps. Note that an MC step is defined as a single MC move, regardless of type and independent of system size.

To run a simulation of 50,000 steps such that thermodynamic quantities are printed every 100 MC steps and coordinates are output every 10,000 steps, use the following:

```
# Frequency_Info
freq_type none
Nthermofreq 100
Ncoordfreq 10000
MCsteps 50000
# Done_Frequency_Info
```

4.1.21 Average

 # Average_Info
Integer(1)

This section specifies how thermodynamic quantities are output. At present, Cassandra writes instantaneous values of thermodynamic quantities at a frequency given by either *Nthermofreq* or *thermofreq* in the `# Frequency_Info` section. *Integer(1)* is set to 1 for this purpose. Later versions of Cassandra will have the ability to output block averages as well. Thus, you will specify the following section in your input file:

```
# Average_Info
1
```

4.1.22 Property Output

 # Property_Info *Integer(i)*
*Character(i,j) * One line for each property*

This section provides information on the properties that are output. More than one section is allowed

for multiple boxes. In this case, each section is separated by a blank line. *Integer(i)* is the identity of the box for which the properties are desired. *Character(i,j)* is the property that is to be output. Each property is specified on a separate line. At present, the acceptable entries include:

`Energy_Total`: Total energy of the system (Extensive) in kJ/mol
`Energy_LJ`: Lennard-Jones energy of the sytem in kJ/mol
`Energy_Elec`: Electrostatic energy of the sytem in kJ/mol
`Energy_Intra`: Total intramolecular energy of the system including bonded and non-bonded interactions in kJ/mol
`Enthalpy`: Enthalpy of the system (Extensive) kJ/mol
`Pressure`: Pressure of the system in bar
`Volume`: Volume of the system in $Å^3$
`Nmols`: Number of mols of species
`Density`: Density of a species in $\#/Å^3$

For example, if you would like total energy, volume and pressure of a one box system to be written, you may specify the following:

```
# Property_Info 1
Energy_Total
Volume
Pressure
```

For a GEMC-NVT simulation, total energy and density of all the species in box 1 and total energy, density of all the species in box 2 along with the pressure may be output using the following format:

```
# Property_Info 1
Energy_Total
Density

# Property_Info 2
Energy_Total
Density
Pressure
```

4.1.23 Fragment Files

 # Fragment_Files
Character(i,j) Integer(i,j) * One line for each fragment i in species j

In this section, information about the fragment library is specified. *Character(i,j)* gives the location of the fragment library of fragment i in species j; *Integer(i,j)* is the corresponding integer id specifying the type of the fragment. This section is automatically generated by library_setup.py. However, if there is a need to change this section, follow the example given below.

For a simulation involving two species of which the first one contains three distinct fragments and species 2

has two identical fragments, this section might look like:

```
# Fragment_Files
frag_1_1.dat 1
frag_2_1.dat 2
frag_3_1.dat 3
frag_1_2.dat 4
frag_1_2.dat 4
```

This will tell Cassandra to use the files frag_1_1.dat, frag_2_1.dat and frag_3_1.dat for the three fragments of species 1. Since species 2 has two identical fragment, Cassandra will use the same fragment library frag_1_2.dat for these fragments.

4.1.24 File Info

 # File_Info
Character(1)

This section is used only while generating a fragment library. Cassandra will use the filename specified in *Character(1)* to store different conformations of the fragment being simulated. Once again, this section is automatically handled by library_setup.py. However, if the user wishes to modify this part, use the following template:

```
# File_Info
frag.dat
```

This will tell Cassandra to store the fragment library in the file named `frag.dat`.

4.1.25 CBMC parameters

 # CBMC_Info
kappa_ins *Integer(1)*
kappa_rot *Integer(2)*
kappa_dih *Integer(3)*
rcut_cbmc *Real(i,4)* * Number of entries equal to number of simulation boxes

Cassandra utilizes a configurational bias methodology based on reservoir sampling [1]. This section sets a number of parameters required for biased insertion/deletion (refer to the sections # Prob_Insertion, # Prob_Deletion and # Prob_Swap) and configurational regrowth (# Prob_Regrowth section) of molecules. For a biased insertion, a fragment is chosen at random and given a random orientation. A number of trial positions are generated for the center-of-mass of the fragment. One of the trial positions is then selected randomly based on the Boltzmann weight of the energy of the trial position. The number of trial insertion positions is given by *Integer(1)*.

Once a trial position for the insertion is chosen, rotational bias may be applied by generating a number of trial orientations. *Integer(2)* specifies the number of such trial orientations. This feature will be imple-

mented in later versions of Cassandra and any value for *Integer(2)*, at present, is ignored. To avoid any confusion, *Integer(2)* is set to 0.

After the biased placement of the first fragment, additional fragments directly bonded to the first fragment are placed. Each of these fragments undergoes a number of trial orientations with respect to the fragment to which it is added. *Integer(3)* controls the number of such orientations that are generated.

For all the trials, energy of the partially grown molecule with itself and surrounding molecules is to be calculated. For this purpose, a short cutoff is used. *Real(i,4)* specifies the cutoff distance in Å for each of the boxes in a simulation. A short cutoff is fast, but might miss some overlaps. You can experiment with this value to optimize it for your system.

For a GEMC simulation in which 12 candidate positions are generated for biased insertion/deletion, 10 trials for biased dihedral angle selection and the cutoff for biasing energy calculation is set to 5.0 Å in box 1 and 6.5 Å in box 2, this section would look like:

```
# CBMC_Info
kappa_ins 12
kappa_rot 0
kappa_dih 10
rcut_cbmc 5.0 6.5
```

4.2 MCF File

A Molecular Connectivity File (MCF) defines the information related to bonds, angles, dihedrals, impropers fragments and non bonded interactions for a given species. One MCF file is required for each species present in the system. The information contained in this file involves the force field parameters, atoms participating in each of the interactions and the functional form used in each potential contribution. The keywords are preceeded by a '#' and comments follow a '!'. Similarly to the input file, the order of the keywords is not important. A complete list of the keywords is provided below.

Note that parameters for all of the following keywords must be separated by spaces only. Do not use the tab character.

Note that MCF files are generated by the script mcfgen.py automatically. The following description is provided for the users who wish to modify the MCF file or build the MCF file on their own.

4.2.1 Atom Info

 # Atom_Info
Integer(1)
*Integer(2) Character(3)*6 Character(4)*2 Real(5) Real(6) Character(7)*20 Real(8) Real(9) Character(10)*

This keyword specifies the information for non-bonded interactions. It is a required keyword in the MCF file. If not specified, the code will abort. The inputs are specified below:

- *Integer(1)*: Total number of atoms in the species.
- *Integer(2)*: Atom index.
- *Character(3)*6*: Atom type up to 6 characters. This string of characters should be unique for each interaction site in the system, i.e. do not use the same atom type for two atoms in the same (or different) species unless the (pseudo)atoms have the same atom types.
- *Character(4)*2*: Atom element name up to 2 characters.
- *Real(5)*: Mass of the atom in amu. Note that for united atom models, this would be the mass of the entire pseudoatom.
- *Real(6)*: Charge on the atom.
- *Character(7)*: Specifies functional form for VDW interactions to be used in the simulation. This must match what is given for `# VDW_Style` (subsection 4.1.4) in the input file. At present only 'LJ' style is permitted.
- *Real(8)*: The energy parameter in K.
- *Real(9)*: Collision diameter (σ) in Å.
- *Character(10)*: Set to 'ring' only if a given atom is part of a ring fragment. Note that a ring fragment is defined as those atoms that belong to the ring (e.g. in cyclohexane, all the six carbons) and any atom directly bonded to these ring atoms (e.g. in cyclohexane, all the hydrogens). In other words, all of the ring and exoring atoms are given the ring flag. For atoms that are not part of rings, leave this field blank.

Note that for species with a single fragment, the branch point atom is listed as the first atom.

For example, for a united atom pentane model:

```
# Atom_Info
5
1 C1_s1 C 15.0107 0.0 LJ 98.0 3.75
2 C2_s1 C 14.0107 0.0 LJ 46.0 3.95
3 C3_s1 C 14.0107 0.0 LJ 46.0 3.95
4 C4_s1 C 14.0107 0.0 LJ 46.0 3.95
5 C5_s1 C 15.0107 0.0 LJ 98.0 3.75
```

The number below the keyword `# Atom_Info` specifies a species with 5 interaction sites, consistent with a united atom pentane model. The first column specifies the atom ID of each of the pseudo atoms. The second and third columns provide the atom type and atom name, respectively. The fourth column represents the atomic mass of each pseudoatom. Note that the mass of C1_s1 is 15.0107 for this united atom model, as it involves a carbon and three hydrogen atoms. The same applies for all other interaction sites. The fifth

column contains the partial charges placed on each of these pseudoatoms. The sixth, seventh and eighth columns contain the repulsion-dispersion functional form, the energy parameter and the collision diameter respectively. In this case, the usual Lennard-Jones functional form is used. Note that none of these atoms used the flag 'ring', as no rings are present in this molecule.

For a molecule containing rings, for example cyclohexane:

```
# Atom_Info
6
1 C1_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
2 C2_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
3 C3_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
4 C4_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
5 C5_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
6 C6_s1 C 14.0107 0.0 LJ 52.5 3.91 ring
```

Note the flag 'ring' was appended as the last column for this cyclic molecule.

Finally, for the SPC/E water model:

```
# Atom_Info
3
1 O1_s1 O 16.00 -0.8476 LJ 78.20 3.1656
2 H2_s1 H 1.000 0.4238 LJ 0.0 0.0
3 H3_s1 H 1.000 0.4238 LJ 0.0 0.0
```

This is a molecule with a single fragment. Therefore, the branch point atom must be specified as the first atom in the list. In this case, oxygen is the branch point and thus its atom ID is 1.

4.2.2 Bond Info

 # Bond_Info
Integer(1)
Integer(i,2) Integer(i,3) Integer(i,4) Character(i,5) Real(i,6) Real(i,7)

This section provides information on the number of bonds in a molecule and atoms involved in each bond along with its type. It is a required keyword in the MCF file. If not specified, the code will abort. The inputs are specified below:

- *Integer(1)*: Total number of bonds in the species. From the next line onwards, the bonds are listed sequentially and information for each bond is included on a separate line.
- *Integer(i,2)*: Index of the i^{th} bond.
- *Integer(i,3) Integer(i,4)*: IDs of the atoms participating in the bond.
- *Character(i,5)*: Type of the bond. At present only 'fixed' is permitted.

- *Real(i,6)*: Specifies the bond length for a particular bond in Å.

Note that at present, Cassandra simulations can be carried out only for fixed bond length systems.

For example, for the water model SPC/E, the `# Bond_Info` section is the following:

```
# Bond_Info
2
1 1 2 fixed 1.0
2 1 3 fixed 1.0
```

In the above example, two bonds are specified whose fixed length is set to 1.0 Å.

4.2.3 Angle Info

 # Angle_Info
Integer(1)
Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) Character(i,6) Real(i,7) Real(i,8)

The section lists the information on the angles in the species. It is a required keyword in the MCF file. If not specified, the code will abort.

- *Integer(1)*: Number of angles in the species.
- *Integer(i,2)*: Index of the i^{th} angle.
- *Integer(i,3) Integer(i,4) Integer(i,5)*: IDs of the atoms participating in the i^{th} angle. Note that *Integer(i,4)* is the ID of the central atom.
- *Character(i,6)*: Type of the angle. Currently, Cassandra supports 'fixed' and 'harmonic' (Eq. section 2.1) angles. For the 'fixed' option, *Real(i,7)* is the value of the angle and *Real(i,8)* is ignored by the code if specified. In the case of 'harmonic' potential type, *Real(i,7)* specifies the harmonic force constant (K/rad^2) while *Real(i,8)* is the nominal bond angle (in degrees).

For example, for a united atom pentane molecule with flexible angles, this section is the following:

```
# Angle_Info
3
1 1 2 3 harmonic 31250.0 114.0
2 2 3 4 harmonic 31250.0 114.0
3 3 4 5 harmonic 31250.0 114.0
```

In the above example, the three angles between the pseudoatoms found in the pentane model are specified. The three angles have an harmonic potential, whose force constant is equal and is set to 31250.0 K/rad^2. Finally, the equilibrium angle for these angles is 114.0°.

An example for SPC/E water model with fixed angles is provided below:

```
# Angle_Info
1
1 2 1 3 fixed 109.47
```

This model has only one angle that is set to 109.47°. Note that this angle is fixed, so there is no force constant.

4.2.4 Dihedral Info

 # Dihderal_Info
Integer(1)
Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) Integer(i,6) Character(i,7) Real(i,8) Real(i,9) Real(i,10) Real(i,11)

This section of the MCF file lists the number of dihedral angles and associated information for a given species. It is a required keyword in the MCF file. If not specified, the code will abort.

- *Integer(1)*: Lists the number of dihedral angles.
- *Integer(i,2)*: Index of the i^{th} dihedral angle.
- *Integer(i,3): Integer(i,6)* - IDs of the atoms in the i^{th} dihedral angle.
- *Character(i,7)* : Dihedral potential type. Acceptable options are 'OPLS', 'CHARMM', 'harmonic' and 'none'. If 'OPLS' dihedral potential type is selected, then the real numbers *Real(i,8) - Real(i,11)* are the coefficients in the Fourier series (see Eq 2.2). The units are in kJ/mol. For the 'CHARMM' dihedral potential type, three additional parameters are specified: a_0, a_1 and δ (section 2.3). If 'harmonic' dihedral potential type is used, then two additional parameters, K_{phi} and ϕ_0 (section Eq 2.4), are specified. For the 'none' dihedral potential type, no additional parameters are necessary.

For example, for a united atom pentane molecule using an OPLS dihedral potential type, the dihedrals are specified as follows:

```
# Dihedral_Info
2
1 1 2 3 4 OPLS 0.0 2.95188 -0.5670 6.5794
2 2 3 4 5 OPLS 0.0 2.95188 -0.5670 6.5794
```

In this model two dihedral angles are specified by atoms 1,2,3,4 and 2,3,4,5. This model uses an OPLS functional form and thus four parameters are provided after the OPLS flag.

4.2.5 Fragment Info

 # Fragment_Info
Integer(1)

Integer(i,2) Integer(i,3) Integer(i,4) Integer(i,5) ... Integer(i,2+Integer(i,3))

This section defines the total number of fragments in a given species. It is an optional keyword. However, if the species is composed of fragments, then this section must be specified. The inputs are specified below:

- *Integer(1)*: Total number of fragments.
- *Integer(i,2)*: Index of the i^{th} fragment.
- *Integer(i,3)*: Number of atoms in the i^{th} fragment.
- *Integer(i,4) ... Integer(i,2+integer(i,3))*: List of the atom IDs in the fragment. The first atom ID is that for the branch point atom. **Atom ordering for the remaining atoms must match the order of atoms in the fragment library files.**

For example, for a pentane united atom model:

```
# Fragment_Info
3
1 3 2 1 3
2 3 3 2 4
3 3 4 3 5
```

This specifies three fragments. Each of these fragments has three atoms. The first atom specified for each of the fragments is the branch point atom.

4.2.6 Fragment Connectivity

 # Fragment_Connectivity
Integer(1)
Integer(i,2) Integer(i,3) Integer(i,4)

The section lists the fragment connectivity - which fragment is bonded to which other fragment. It is a required keyword if **Fragment_Info** is specified.

- *Integer(1)*: total number of fragment connections.
- *Integer(i,2)*: index of the i^{th} fragment connectivity.
- *Integer(i,3) Integer(i,4)*: fragment IDs participating in the connectivty.

For example, for a pentane united atom model:

```
# Fragment_Connectivity
2
1 1 2
2 2 3
```

In this example, there are three fragments, therefore, two fragment connectivities must be specified. Note that fragment 1 is connected to fragment 2 and fragment 2 is connected to fragment 3.

Bibliography

[1] J. K. Shah and E. J. Maginn. A General And Efficient Monte Carlo Method for Sampling Intramolecular Degrees of Freedom of Branched and Cyclic Molecules. *J. Chem. Phys.*, 135:134121, 2011.

[2] D. Fincham. Optimization of the Ewald Sum For Large Systems. *Mol. Sim*, 13:1–9, 1994.

[3] P. L'Ecuyer. Tables of Maximally Equidistributed Combined LFSR Generators. *Math. Comput.*, 68:261–269, 1999.