EMC: Monte Carlo Simulations

A General Simulation Package
EMC version 9.4.4, 1 August, 2023

Pieter J. in 't Veld

This manual is for EMC (version 9.4.4, 1 August, 2023), a simulation package that can build and simulate both atomistic and coarse-grained systems using Monte Carlo techniques.

Table of Contents

1 Introduction

1.1 General Introduction

Monte Carlo simulation techniques are wide spread through branches of scientific research and financial industry. It finds its application in statistical sampling of complex functional descriptions, giving numerical solutions to theoretically unsolvable equations. Typically codes using Monte Carlo techniques are written with a specific purpose in mind. These approaches include scientific implementations where versatility and not expandability is the main design goal. The latter finds an example in Towhee, a code which offers a wide array of force fields and sampling techniques, but encounters challenges when the need arises for implementation of new techniques. The code presented here does not try to offer the same breath, but rather follows a strategy of expandability. Strict modular design isolates key features in separate objects, yet a general parser allows for easy integration of these objects in a coherent design, thus allowing for a more flexible and expandable design. The purpose of this paper is to present a design philosophy for simulation engines in general, while applying this philosophy to the field of scientific Monte Carlo simulations. This is achieved by the identification of key features, such as input/output features, force field calculation, and configurational alterations, features which are highlighted in later sections.

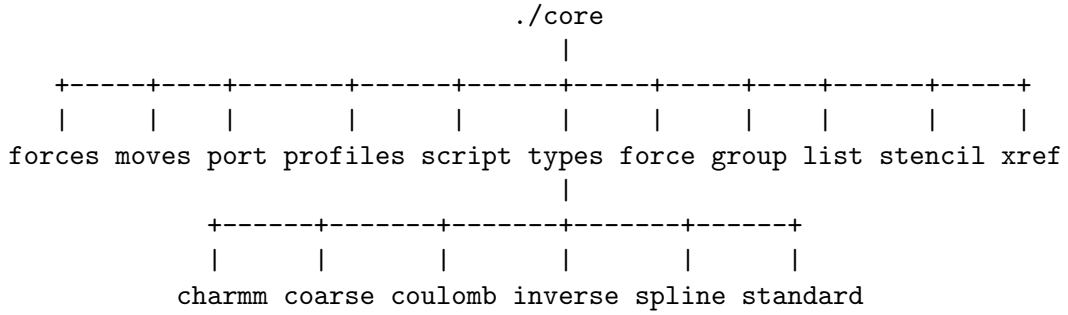
Each simulation code hinges around computational efficiency. Gain in speed can be obtained by an increase in processor speed, but more importantly, by judicious design of the program framework. This requires the identification of key factors of importance. Specifically, computational efficiency for this type of simulations is hampered by its serial nature: atoms or sites are moved one after another and not in a parallel fashion as is the case with Molecular Dynamics. Efficiency of energy calculations becomes the major concern when designing Monte Carlo codes for scientific applications. These applications hinge around the concept of an acceptance rule, in which typically is formed by a combination of system temperature with the difference in energy as the result of a change in configuration. This configuration change or move normally entails the displacement of one or a few sites. The serial nature of these moves necessitates the calculation of configurational energy both before and after the move. Optimization of energy calculations therefore plays an integral role in improving computational efficiency, for which this paper offers suggestions.

Alternatively, the speed with which a scientific field is allowed to advance rests on the foundations on which new developments can be build. Normally, code optimization results in an increase in code complexity, which in turn creates barriers for non-professional programmers to implement alterations necessary to test new scientific ideas. Strict separation of modules, yet flexible integration of concepts, creates a library-type of environment in which non-professional programmers – with the help of simple development tools – can use already existing modules as templates for their new ideas, thus creating a canvas on which only the minimal necessary changes need to be made.

1.2 Distribution Content

This distribution consist of a set of hierarchical directories, each of which represents a part of the code or documentation. The main directories are `./core`, `./emc`, `./examples`, `./scripts`, `./modules`, `./lib`, and `./texinfo`. A general make file `./Makefile` governs

compilation of separate pieces of this distribution. A '`make`' in the root directory displays all compilation options. General routines are grouped under `./core`, which has the following structure



Each subdirectory represents a substructure of main structures. Any module or object files placed in any of the directories are automatically compiled into each executable, provided that they conform to a required format associated with the substructure. Definition of macros based on file headers facilitates automatic compilation. This particular feature aids programming by example, which means an already existing module can be used as a template, rather than having to completely write a new model. A copy script to facilitate this process is provided by `./scripts/cpmod.pl`.

Currently, the main program, which includes a scripting front-end, is located in `./emc`. This directory also contains example building and running scripts in `./examples`. Upon compilation through execution of '`make emc`' generates a new directory `./emc/objects`, which contains all `.o` files. Examples in `./examples` make use of predefined crystal structures in `./lib`.

All documentation is organized in `./texinfo`. This PDF document is generated by a '`make pdf`' in this directory.

2 Methodology

2.1 Lists

The presented code employs a combination of cell lists and verlet lists, where a separate object represents each list. Spatial discretization forms the cell list basis and is determined by either the largest or the smallest force field cut off, combined with a user supplied skin. The choice of cut off depends on invoking a conventional cell list, in which case the largest cut off is selected, or a multi-neighbor cell list, in which case the smallest cut off is chosen (*Intveld2008*). Both lists store their members in a linked list, which is accomplished by the application of a stencil or template. This template describes which cells neighbor the central cell. A distance criterion – based on the cut off and skin – then decides which sites are eventually added to a verlet list. Besides an integer vector to neighboring cells, a multi-neighbor template also contains the minimum distance between the cells containing the each site contributing to the pair interaction, which enables a primary selection of sites based on their cut off and therefore negating the initial need of a distance calculation. Inclusion in a verlet list only then requires a distance calculation when the neighboring particle falls within the stored minimum distance. Application of the multi-neighbor principle is optional, although initial tests do not indicate a significant drop in performance compared conventional cell list implementations.

Verlet lists describe which neighboring sites fall within a certain distance criterion of a site of interest. These lists are described by linked lists. Cross-linking between verlet lists has been added to allow for easy removal of a site when performing a Monte Carlo move. Practically, this means that each verlet list member of one site knows which verlet list member of another site it is connected to, thus creating members that contain pointers to child and parent in one particle's list and a pointer to a member of another particle's list. Additionally, the squared distance between the two sites has been added to negate the need for distance calculations during non-bonded force field evaluation.

Typically, the application of verlet lists in Molecular Dynamics makes use of the principle of a skin, which is a distance added to the force field cut off, and serves the purpose of reducing the number of verlet list rebuilds. However, an increase in distance calculations, as a result of inclusion of more sites in the verlet list, forms the downside of skin application. Hence, a balance needs to be found between performance gain as a result from less verlet list builds and performance loss as a result of an increase in distance calculations. Typical numbers used for the skin are 0.3σ in Lennard-Jones (LJ) units or 1 $\mathring{A}$ in atomistic units. An added advantage of the usage of skins is that the decision to rebuild a verlet list can be coupled to the distance with which a site has moved from when the verlet list was built last for that site. A safe criterion is to rebuild the list when the site has moved by more than half the skin. Application of this criterion enables a dynamic update of verlet lists based on particle positions only and not by an external constraint such as a preset number of passed time steps or cycles, as is commonly used in packages such as LAMMPS (*Plimpton1995*).

2.2 Force Fields

Implemented force fields include standard representations of the OPLS force field as described by Jorgensen et al. (*Jorgensen*), the CHARMM force field as described by MacKerell

et al. (*MacKerell1998*), and a Class 2 force field representation used by PCFF or COMPASS force fields as described by Sun (*Sun1998*). A separate module describes coulombic interaction and allows for the use of either a cut off or Ewald summations to account for the long-range character (*Frenkel*). Coarse-grained force fields are provided by a colloidal force field as described by Everaers and Etjehadi (*Everaers2003*), which includes additional cross-terms for interaction between Lennard-Jones and colloidal particles (*Grest2007*, *Veld2008*), and the FENE bond potential as described by Kremer and Grest (*Kremer*). Pressures are calculation through standard virial expressions, which were derived when needed. All interactions can be compounded without significant effects on performance.

Generalization of virial calculations requires a solution to the gradient of a potential with respect to the cartesian coordinate system, which entails splitting partial derivatives into a product of linear parts and cartesian parts. For a nonbonded potential dependent on radius r this results in

$$\frac{\partial U(r)}{\partial \vec{x}} = \frac{\partial U(r)}{\partial r}\frac{\partial r}{\partial \vec{x}} = \frac{\partial U(r)}{\partial r}\frac{\vec{x}}{r},$$

where $\vec{x}$ represents the constituent vector of a cartesian axis frame. Consequently, the virial tensor W_r resulting from this expression becomes

$$W_r(r_{ij}) = -\frac{\partial U(r)}{\partial r} r_{ij} \Delta\vec{r}_{ij} \Delta\vec{r}_{ij}.$$

where r_{ij} denotes the distance between site i and site j and $\Delta\vec{r}_{ij}\Delta\vec{r}_{ij}$ denotes a vector product resulting in a tensor. Bonded potentials potentials depend on bond length l_{ij}, angle θ_{ijk}, torsion ϕ_{ijkl}, and improper ψ_{ijkl}, which are defined as

$$\Delta\vec{r}_{ij} = \frac{\vec{r}_j - \vec{r}_i}{|\vec{r}_j - \vec{r}_i|}$$

$$\vec{n}_{ijk} = \frac{\Delta\vec{r}_{jk} \times \Delta\vec{r}_{ij}}{|\Delta\vec{r}_{kj} \times \Delta\vec{r}_{ji}|},$$

$$r_{ij} = l_{ij} = |\vec{r}_j - \vec{r}_i|,$$

$$\theta_{ijk} = \Delta\vec{r}_{ij} \cdot \Delta\vec{r}_{jk},$$

$$\phi_{ijkl} = \vec{n}_{ijk} \cdot \vec{n}_{jkl},$$

$$\psi_{ijkl} = \vec{n}_{ijk} \cdot \vec{n}_{ijl}.$$

Note that j represents the center site of impropers. The bond virial tensor W_l for potentials depending on l_{ij} is described by

$$W_l(l_{ij}) = -\frac{\partial U(l_{ij})}{\partial l_{ij}} l_{ij} \Delta\vec{r}_{ij} \Delta\vec{r}_{ij}.$$

The angular virial tensor W_θ for potentials depending on θ_{ijk} is described by

$$W_\theta(\theta_{ijk}) = -\frac{\partial U(\theta_{ijk})}{\partial \theta_{ijk}} \frac{1}{\sin(\theta_{ijk})} \left[\Delta\vec{r}_{ij} \left(\Delta\vec{r}_{jk} + \cos(\theta_{ijk})\Delta\vec{r}_{ij}\right) + \Delta\vec{r}_{jk} \left(\Delta\vec{r}_{ij} + \cos(\theta_{ijk})\Delta\vec{r}_{jk}\right)\right].$$

Note that vector product $\Delta\vec{r}_{ij}\Delta\vec{r}_{jk}$ is noncommunicative, i.e. $\Delta\vec{r}_{ij}\Delta\vec{r}_{jk} \neq \Delta\vec{r}_{jk}\Delta\vec{r}_{ij}$. The torsional virial tensor W_ϕ for potentials depending on ϕ_{ijkl} is described by

$$W_\phi(\phi_{ijkl}) = -\frac{\partial U(\phi_{ijkl})}{\partial \phi_{ijkl}} \frac{1}{\sin(\phi_{ijkl})} [$$

$$(\cos(\phi_{ijkl})\Delta\vec{r}_{jk} \times \vec{n}_{ijk} + \Delta\vec{r}_{jk} \times \vec{n}_{jkl})\, \Delta\vec{r}_{ij}$$
$$+ (\cos(\phi_{ijkl})\, (\Delta\vec{r}_{kl} \times \vec{n}_{jkl} - \Delta\vec{r}_{ij} \times \vec{n}_{ijk}) + \Delta\vec{r}_{kl} \times \vec{n}_{ijk} - \Delta\vec{r}_{ij} \times \vec{n}_{jkl})\, \Delta\vec{r}_{jk}$$
$$- (\cos(\phi_{ijkl})\Delta\vec{r}_{jk} \times \vec{n}_{jkl} + \Delta\vec{r}_{jk} \times \vec{n}_{ijk})\, \Delta\vec{r}_{kl}] .$$

The improper virial tensor W_ψ for potentials depending on ψ_{ijkl} is described by

$$W_\psi(\psi_{ijkl}) = -\frac{\partial U(\psi_{ijkl})}{\partial \psi_{ijkl}} \frac{1}{\sin(\psi_{ijkl})} [$$

$$(\cos(\psi_{ijkl})\Delta\vec{r}_{jk} \times \vec{n}_{ijk} + \Delta\vec{r}_{jk} \times \vec{n}_{ijl})\, \Delta\vec{r}_{ij}$$
$$+ \cdots$$
$$+ (\cos(\psi_{ijkl})\Delta\vec{r}_{jk} \times \vec{n}_{ijl} + \Delta\vec{r}_{jk} \times \vec{n}_{ijk})\, \Delta\vec{r}_{kl}] .$$

3 Program Structure

This particular Monte Carlo simulation code is modular in approach. While it uses C as its programming language foundation, it incorporates distinct object oriented features. Communication by all modules to the outside, it being storage or inter-node communication, happens through a generalized parsing module, which can be used for both textual and binary output. Binary output can be either to memory or file. Variables are stored in a tree structure of pointers with cross-links between branches when so required. The textual input/output file bears an identical but simplified structure compared to the tree structure in memory: communication only conveys essential information and cross-links are implied. The structure 'simulation' presents the top structure, consisting of pointers to the structures 'io', 'identity', 'clocks', 'output', 'units', 'channels', 'types', 'systems', 'moves', 'statistics', 'forces', 'profiles', 'clusters', 'groups', and 'sites'. These structures are grouped in the following categories:

- Molecular representation by 'clusters', 'groups', and 'sites'
- Molecular interaction by 'types' and 'forces'
- Simulation parameter definition by 'systems'
- Configuration changes by 'moves'
- Measurements by 'statistics' and 'profiles'
- Communication by 'io', 'output', and 'units', and 'channels'
- Simulation timing by 'clocks'
- Simulator identification by 'identity'

3.1 Molecular Representation

A need for generality motivates the chosen terminology of 'clusters', 'groups', and 'sites'. Clusters can represent molecules and generally define closed networks of connected sites. Groups are of the order of repeat unit or protein residue and generally define subsets of molecules, creating the possibility of group operators when applying configuration changes. Sites can resemble atoms, but could also represent a more coarse-grained model of a group of atoms lumped into one site. Sites are the backbone of all interactions. This section describes the hierarchy from the bottom up, starting with 'sites', followed by 'groups' and 'clusters'.

3.1.1 Sites

Description of sites.

3.1.2 Groups

Description of groups.

3.1.3 Clusters

Description of clusters.

3.2 Molecular Interactions

3.2.1 Types

Description of types.

3.2.2 Forces

Description of forces.

3.3 Systems

Description of systems.

3.4 Configurational Moves

Description of moves.

3.5 Measurements

Description of distributions.

4 Simulation Setup

4.1 General

EMC provides a setup script `emc_setup.pl`, which allows for creation of EMC build and LAMMPS input scripts. The EMC setup script serves as a wrapper around EMC itself, creating a context for abstraction of EMC scripting commands. In its simplest appearance, setup scripts only contain a few lines describing simulated chemistry and their quantities, where subsequent options are defined through the command line. More complex scripts contain several paragraphs for describing simulation conditions and chemistry. This mode will be referred to as the chemistry mode (see Section 4.6.2 [Chemistry File], page 38). The most complex mode creates an environment in which several simulations are defined in one script, allowing for looping over multiple simulation conditions. This mode will be referred to as the environment mode (see Section 4.6.1 [Environment File], page 35). EMC environment setup scripts function as a wrapper around chemistry scripts. An environment setup script can hold multiple chemistry scripts. Paragraphs use keyword `ITEM` as demarcation, where identifiers follow this keyword at the beginning of each paragraphs. Their ending is marked by `ITEM END`.

Normal parametrization uses force fields as provided by EMC (see Section 5.11 [Field], page 106) for typing. These force fields can be found in `${EMC_ROOT}/field/`. Optional `references.csv` and `parameters.csv` files can be provided for DPD force fields. The latter is called from the chemistry file. Alternatively, a more general way in provided by using the field paragraph (`ITEM FIELD`) in an environment script. Both produce a force field file with extension `.prm`. Force fields are governed over `types` definitions (see Section 7.12 [Types], page 204). The EMC setup script can create multiphase initial configuration, but does not adress the grafting capabilities of the EMC build capabilities. The EMC setup script displays all available options when called without arguments. It is, in general, called by

```
emc_setup.pl [-option[=#]] project [phase 1 clusters [+ ...]]
```

It can also, however, be used as an interpreter, where EMC setup scripts can function as executables when using EMC in a UNIX-type environment. An interpreter allows for including all settings and options into an EMC script, thus folding the description of one simulation or a set of simulations into one comprehensive script.

4.2 Setup Usage

Examples can be found in `${EMC_ROOT}/examples/setup/`. Creating EMC and LAMMPS input with these examples occurs in steps. The first step is to convert the example setup script by means of `emc_setup.pl` through

```
emc_setup.pl example.esh
```

which either creates an input script for EMC, called `build.emc` by default, and an input script for LAMMPS, called `example.in` by default, or sets up an execution environment by creating subdirectories `build`, `run`, `analysis`, and `chemistry`. The first three directories contain `bash` scripts with the purpose of respectively building input structures, running molecular dynamics simulations, and analysis of end results. The `chemistry` directory can contain force fields, variation in chemistry, polymer make up, etc. Default names can be altered when desired (see ⟨undefined⟩ [Enviroment Options], page ⟨undefined⟩).

The second step involves a call of EMC itself, by for instance for LINUX,

```
emc_linux build.emc 2>&1 | tee build.out
```

The latter part of the command allows for writing output to screen and `build.out` simultaneously. Five extra files appear upon completion of this command, i.e. `example.pdb`, `example.psf`, and `example.vmd` for visualization with VMD, and `example.data` and `example.params` for use with LAMMPS. Visualization supports checking the validity of the final structure. The call to VMD, when in the user's path, is facilitated by,

```
vmd -e example.vmd
```

The `example.vmd` file represents a small Tcl script, which contains information about particle diameters (as defined by the chosen force field) and the type of VMD presentation to use for visualization. A molecular dynamics simulation with LAMMPS occurs with for example

```
lmp_linux <example.in 2>&1 | tee example.out
```

The `example.in` LAMMPS script contains all the necessary information for executing the run. Please note, that the presented examples in the `${EMC_ROOT}/examples/setup/` directory are primarily meant as building examples. Strategies for managing sets of simulations are provided by the environment mode of `emc_setup.pl` (see Section 4.6.1 [Environment File], page 35), for which an example can be found in `${EMC_ROOT}/examples/setup/shear/`.

4.3 Extensions

A comprehensive mode for EMC Setup is when using environment scripts, which creates a modeling environment consisting of build, run, and analysis scripts, complemented with a hierarchical structure for dealing with produced imulation data. The simpler EMC chemistry script creates a subset of the more comprehensive environment script, but is useful for prototyping simulations. Extensions an explanations for scripts and data files produced by both modes following from EMC Setup, EMC, LAMMPS or postprocessing analysis can be found in the following table,

Extension	Description
`.data`	EMC output containing LAMMPS data file defining coordinates, types, and topology
`.csv`	Postprocessed data derived from EMC or LAMMPS output resulting from execution of EMC analysis scripts; normally used for further processing by user generated scripts (using e.g. Python, Perl, Matlab, or Mathematica)
`.density`	Density profile produced by LAMMPS
`.density3d`	3D density profile produced by LAMMPS
`.emc`	EMC script as interpreted by EMC itself when produced by `emc_setup`; used to create input structures by means of energetic considerations; EMC produces a comprehensive EMC format with the same extension as output containing all topological and morphological information
`.energy`	Time-averaged energetic information produced by LAMMPS
`.esh`	EMC Setup script as interpreted by wrapper `emc_setup.pl`; can be either a chemistry or an environment script
`.in`	LAMMPS input script according to specifications as made in a EMC chemistry file
`.params`	EMC output following from a successful build; input needed by LAMMPS, containing all paramaters defining a full simulation
`.pdb`	PDB format containing initial coordinates
`.pressure`	Time-averaged pressure information produced by LAMMPS
`.prm`	EMC parameter file as produced by either EMC chemistry or environment script; used by EMC to set force field energetics

`.psf`	PSF format containing types and (initial) topology
`.vmd`	VMD Tcl script used to read in both `.pdb` and `.psf`, additionally defining bead sizes following from the used force field
`.volume`	Time-averaged volume information produced by LAMMPS

4.4 Enviroment Options

Option	Default	Description
`analyze_archive`	`true`	Toggle archiving of file names associated with analyzed data; file name lists can be used directly or at a later date for transferring data between computational clusters and local machines.
`analyze_data`	`true`	Toggle creation of tar archive from exchange file list; this data is used in combination with ./scripts/run_host.sh -exchange to transfer data from computational clusters to local machines; archives should not be created when analysis scripts subsequently submit analysis to queue.
`analyze_last`	`false`	Toggle inclusion of the last trajectory frame during analysis (deprecated); analysis is transfered to the analysis paragraph (see Section 4.6.1 [Environment File], page 35).
`analyze_replace`	`true`	Toggle replacement of already exisiting analysis results.
`analyze_skip`	`0`	Set the number of initial frames to skip during analysis.
`analyze_source`	-	Specify alternate data source directory for analysis scripts.
`analyze_user`	-	Set directory for user analysis scripts, which can be accessed through an analysis paragraph (see Section 4.6.1 [Environment File], page 35).
`analyze_window`	`1`	Set the number of frames over which to apply a window averaging during analysis.
`modules`	-	Manipulate runtime modules; format is defined by `[command]=module` separated by commas.
`name_analyze`		Set the name of job analysis scripts; analysis scripts are stored in `./analyze/`; takes the base name of the chemistry script when omitted; ignored when `"-"` is chosen as name.

`name_build`		Set the name of job build scripts; build scripts are stored in `./build/`; takes the base name of the chemistry script when omitted; ignored when `"-"` is chosen as name.
`name_run`		Set the name of job run scripts; run scripts are stored in `./run/`; takes the base name of the chemistry script when omitted; ignored when `"-"` is chosen as name.
`name_scripts`	-	Simultaneously set analysis, build and run script names.
`name_testdir`	-	Set the name of a test directory in which to test the chemistry paragraph in an environment script; a setup script is created in `./test/`; ignored as default.
`nchains`	-	Set the number of chains to be used for executing LAMMPS jobs; allows for having the next job wait on the previous (currently only with LSF).
`ncores`	-	Set the number of cores for execution of LAMMPS jobs.
`ncorespernode`	`default`	Set number of cores per node for execution of packed jobs; equivalent to `queue_ppn`.
`preprocess`	`false`	Apply C preprocessing to allow for programmatic flow when interpreting templates; C preprocessing start at the beginning of a line and are preceeded by a `#`-mark; allowed options are `define`, `if`, `elif`, `else`, and `endif`.
`project`	-	Set the project name; derived from setup script name when not given.
`queue`	`account=none`, `analyze=default`, `build=default`, `memory=default`, `ncores=-`, `ppn=default`, `run=default`, `user=none`	Directly set queueing system settings, where option `account` sets billing account, `analyze` sets the analysis queue, `build` sets the build queue, `memory` sets the amount of memory needed, `ncores` sets the number of cores for running, `ppn` sets the number of processors per node, `run` sets the run queue, and `user` sets user options; the latter is dependent on the installed queueing system.

`queue_account`	`none`	Set queue account for billing.
`queue_analyze`	`default`	Set the queue used in analysis scripts; option `default` indicates the use of queue, which is set as system default; option `local` will sequentially execute all jobs on local machine.
`queue_build`	`default`	Set the queue used in build scripts; option `default` indicates the use of queue, which is set as system default; option `local` will sequentially execute all jobs on local machine.
`queue_memory`	`default`	Set memory per core in gb for executed jobs.
`queue_ncores`	`-1`	Set the number of cores for execution of MD jobs.
`queue_ppn`	`default`	Set cores per node for executed jobs.
`queue_run`	`default`	Set the queue used in run scripts; option `default` indicates the use of queue, which is set as system default; option `local` will sequentially execute all jobs on local machine.
`queue_user`	`none`	Options to be passed directly to queuing system; allowed option are dependent on the installed queueing system.
`quiet`	-	Supress all output generated by this setup script.
`replace`	`false`	Replace or overwrite all written script files.
`time_analyze`	`00:30:00`	Set the desired wall time for analysis scripts.
`time_build`	`00:10:00`	Set the desired wall time for build scripts.
`time_run`	`24:00:00`	Set the desired wall time for run scripts.
`workdir`	-	Set the work directory, which is used as a base for creating the EMC work environment.

4.5 Chemistry Options

Option	Default	Description
angle	5,180	This option can be used to set angle constants in three ways: 1. set DPD angle constants (k, theta0), 2. provide additional angle table entries through type1, type2, type3, k, theta0, 3. set force field handling for angle typing; valid options are ignore, complete, warn, empty, and error.
auto	false	Include wildcard mass entry in generated DPD .prm force field file; options are true or false.
binsize	0.01	Set the general bin size to be used in LAMMPS profiles; note, that profiles are recorded in reduced units, i.e. coordinates run from 0 to 1.
bond	25,1	Set DPD bond constants (k, l0); can also be used to provide additional bond table entries through -bond=type1,type2,k,l0.
build	build	Set build script name.
build_center	false	Place first site at the origin (see also option build_origin); options are true or false.
build_dir	../build	Set location of build directory for LAMMPS script.
build_order	false	Build clusters in the order as defined by option phases; options are random or sequence.
build_origin	x=0, y=0, z=0	Set alternate origin at which to place te first site.
build_replace	false	Toggle replacement of already existing build scripts and results, which will skip build execution when a LAMMPS data file exists.
build_theta	false	Set the minimal insertion angle
charge	true	Indicator for occurence of charges in chemistry.csv.
charge_cut	9.5	Set pairwise charge interaction cut off.

`chunk`	`true`	Depricated; use chunk approach for computing profiles and samples in LAMMPS script; options are `true` or `false`.
`communicate`	`false`	Depricated; use communicate keyword instead of comm_modify in LAMMPS input script; used with DPD simulations for backwards compatibility with older versions of LAMMPS; options are `true` or `false`.
`core`	`-1`	Set particle core diameter; used with Born potentials.
`cross`	`false`	Include nonbond cross terms in LAMMPS params file; default depends on chosen force field; options are `true` or `false`.
`crystal`	`false`	Let EMC derive a crystal structure based on the offered import; options are `true` or `false`.
`cut`	`9.5`	Set pairwise interaction cutoff; options are either definite positive numbers or `repulsive`; the latter transfers the the parameters for a purely repulsive cut and shift potential to LAMMPS

`cutoff`	`center=-1,`
	`charge=9.5,`
	`ghost=-1,`
	`inner=-1,`
	`outer=-1,`
	`pair=9.5,`
	`repulsive=0,`
	`rmax=-1`
		Set various cutoffs; options are either definite positive numbers; negative numbers refer to internal defaults; the different cutoffs can also be set with single options charge_cut, ghost_cut, inner_cut, outer_cut, cut, and rmax.

`debug`	-	Turn debugging information on; options are `true` or `false`.

`deform`	`nblocks=1,`
	`ncycles=100,`
	`type=relative,`
	`xx=1, yy=1,`
	`zz=1, zy=0,`
	`zx=0, yx=0`
		Invoke affine deformation using Monte Carlo relaxation when `ncycles` is larger than zero; defomation `type` can either be `absolute` or `relative`.

`delete`	`phase=1, fraction=1.0, thickness=1, type=relative, mode=include, sites=all, groups=all, clusters=all`	Defines the selection criteria for deleting clusters at the edges of the box; `phase` sets the phase to which to apply deletion; `fraction` currently is not used; `thickness` sets the thickness of the region taken into consideration; `type` sets the units of the thickness (valid options are `absolute` and `relative`); `mode` is currently not used; `sites` sets the sites to include in the selection (valid options are `all` for all available sites or a selected list of site types separated by colons); `groups` sets the groups to include in the selection (valid options are `all` for all available groups or a selected list of group ids separated by colons); `clusters` sets the clusters to include in the selection (valid options are `all` for all available clusters build in the set `phase` or a selected list of cluster ids separated by colons); only full clusters are deleted, even when only one site of these clusters complies with the set selection.
`density`	`1[,...]`	Set simulation density; density of separate phases can be entered through separating values for each phase by commas; units and default depend on chosen force field `type`.
`depth`	`auto`	Set the depth with which rings - as defined in groups - are recursively assigned; options are either `auto` or values larger than 2.
`dielectric`	`0.2` or `1`	Set dielectric constant of medium; default depends on chosen force field.
`direction`	`x`	Set direction in which phases in setup scripts with multiple phases are build; valid options are `x`, `y`, or `z`.
`dtdump`	`100000`	Depricated; frequency with which LAMMPS adds to trajectory files.
`dtrestart`	`100000`	Depricated; frequency with which LAMMPS writes restart files.
`dtthermo`	`1000`	Depricated; frequency with which LAMMPS generates thermodynamical output.

`emc`	`true`	Create EMC build script (.emc file); options are `true` or `false`.
`emc_depth`	`8`	Set ring recognition depth in groups paragraph; this option sets the maximum occuring ring size; valid input consists of a positive integer.
`emc_exclude`	`build=false`	Exclude specified sections from the resulting EMC script; excluding `build` omits morphology generation, which can be desired when exporting SMILES.
`emc_execute`	`-`	Execute the EMC build script as created by the setup script; options `false` or - do not execute; option `true` allows EMC setup decide which EMC version to use, based on operating system, host option, and path; any other option value is interpreted as the location and name of the EMC version to use.
`emc_export`	`smiles=false`	Add export of specified format; `smiles` controls export of SMILES for all built clusters in both group-based and site-based representations;option `false` deselects export; other valid options are `csv`, `json`, and `math` for comma separated values, JSON, and Mathematica formats respectively.
`emc_moves`	`displace=1`	Control frequency of selected Monte Carlo move for positive values; zero or negative values unselect.
`emc_output`	`debug=false, exit=true, info=true, warning=true`	Control output as generated by EMC scripts; options for each separate keyword are `true` or `false`.
`emc_progress`	`build=true, clusters=false`	Control progress indicators resulting from executing EMC script; keyword `build` controls progress output during build; keyword `cluster` controls progress output during sequencing and cluster construction; options for each separate keyword are `true` or `false`.

`emc_run`	`nblocks=100, ncycles=0, nequil=0, clusters=all, groups=all, sites=all`	Execute a Monte Carlo simulation after building for `ncycles`; optionally, an equilibration phase is selected when `nequil` is larger than zero; `nblocks` selects the frequency with which output is generated; specific selections can be set when defining either `clusters`, `groups`, or `sites` separately or in combination; `all` indicates that all members of a selection are considered.
`emc_test`	`false`	Test the validity of an EMC environment script; checks up to execution of EMC build files, as generated by EMC environment; options are `true` or `false`.
`emc_traject`	`append=true, frequency=0`	Add generation of trajectory file during Monte Carlo run when `frequency` is larger than zero.
`environment`	`false`	Interpret EMC shell script (.esh) as an environment script; options are `true` and `false`.
`ewald`	`true`	Switch the electrostatics long-range treatment through Ewald summations on or off; default depends on the chosen force field; options are `true` or `false`.
`exclude`	`true`	exclude previous phase during build process; options are `wall`, `soft`, `true`, or `false`; `wall` places a wall between each created phase, avoiding separate phases blending into neighboring phases; `soft` and `true` allow for edges of phases to blend into neighboring phases; `false` does not invoke any restrictions.
`expert`	`false`	Invokes expert mode, which allows for overrides of certain consistency checks (e.g. nrepeat for polymers); options are `true` or `false`.
`extension`	`.esh`	Set the extension of environment scripts; extensions are derived from given environment scripts, when not defined.
`field`	`opls`	Set force field type and name based on root location; more general access to EMC provided force fields uses the `type` option below.

`field_angle`	-	Set error handling for typing of angles; valid options are `complete`, `empty`, `error`, `ignore`, and `warn`.
`field_bond`	-	Set error handling for typing of bonds; valid options are `complete`, `empty`, `error`, `ignore`, and `warn`.
`field_charge`	`true`	Check system charge after applying force field; valid options are `true` and `false`.
`field_check`	`true`	Check force field compatibility upon loading of multiple force fields (i.e. for CHARMM); valid options are `true` and `false`.
`field_debug`	`false`	Set force field debugging options; valid options are `false`, `true`, `reduced`, and `full`, whereby the latter two correspond to reduced and full output respectively.
`field_dpd`	`auto=false, bond=false`	Set options influencing the generation of a DPD force field; possible options are `auto` and `bond` with values of either `true` or `false`; `auto` controls the addition of wildcard additions to mass and nonbonded force field paragraphs; `bond` allows for verbatim transcription of pair to bond interactions.
`field_debug`	`false`	Set debug option, showing individual typing steps; valid options are `full`, `reduced`, and `false`; `full` outputs verbose debugging information, `reduced` generates an abridged selection, and `false` turns debugging information off.
`field_error`	`true`	Exit upon error (set to `false` when debugging a force field).
`field_format`	`%15.10e`	Sets the format of parameters in the produced field; should be format as is standard for floating point values when used in combination with `printf()`.
`field_group`	-	Set field group option for applying force fields to groups as defined by their `TEMPLATE` section.
`field_id`	-	Set force field id.

`field_improper`	-	Set error handling for typing of impropers; valid options are `complete`, `empty`, `error`, `ignore`, and `warn`.
`field_increment`	-	Set error handling for typing of bond increments; valid options are `complete`, `empty`, `error`, `ignore`, and `warn`.
`field_location`	-	Provide force field location; intended for expert use; use `field_type` for EMC provided force fields.
`field_name`	-	Provide force field name; intended for expert use= use `field_type` for EMC provided force fields.
`field_nbonded`	`0`	Define the number of sites, that are bonded, but should be excluded from nonbond interactions, e.g. a value of `1` excludes 1-2 interactions, a value of `2` excludes 1-2 and 1-3 interactions, etc.; allowed values are larger or equal than `0`; this option only applies to DPD force fields.
`field_reduced`	`false`	set force field reduced units flag; valid options are `true` and `false`.
`field_torsion`	-	Set error handling for typing of torsions; valid options are `complete`, `empty`, `error`, `ignore`, and `warn`.
`field_type`	`opls`	Provide force field type; invoking this option will automatically set `field`, `field_location`, and `field_name` options.
`field_write`	`true`	Interpret and write the force field as defined in the `ITEM FIELD` paragraph (see Section 4.6.3 [Field File], page 48)
`focus`	-	List of clusters to focus on.
`grace`	`0.9999, 0.9999,0`	Depreciated command, use `weight` instead; user provided grace for building procedure; given in order of nonbonded, bonded, and focussed interactions.
`ghost_cut`	-	Set molecular dynamics ghost region size for property communication.

help	-	Display script help; option module adds originating modules to the help output.
host	-	Set host on which to run EMC and LAMMPS.
info	-	Turn on information during setup script execution.
inner	-	Set inner cutoff; use and default depend of chosen force field; note that inner and outer cutoffs are interpreted as fractions for colloidal force fields.
insight	true	Create InsightII CAR and MDF output files.
insight_ compress	true	Control compression of produced CAR and MDF files; options are true or false.
insight_pbc	true	Toggle application of periodic boundary conditions; options are true or false.
insight_unwrap	clusters	Set CAR unwrapping mode; options are none, clusters, or sites.
kappa	1 or 4	Set long range electrostatics Ewald summation kappa; default depends on chosen force field; the provided number needs to be positive definite.
lammps	true	Create LAMMPS input script (.in file) or set lammps version using year, e.g. -lammps=2014 (new versions start at 2015) or keywords old or new; alters the settings for -communicate and -chunk; default is new.
lammps_chunk	true	Use chunk approach for computing profiles and samples in the input script; options are true or false.
lammps_ communicate	false	Use communicate keyword instead of comm_modify in the input script; used with DPD simulations for backwards compatibility with older versions of LAMMPS; options are true or false.
lammps_cutoff	false	Toggle output of pairwise cut off in parameter file; valid options are true or false.
lammps_dlimit	0.1	Set nve/limit distance used during equilibration.

`lammps_dtdump`	`100000`	Frequency with which adds to trajectory files.
`lammps_dtrestart`	`100000`	Frequency with which writes restart files.
`lammps_dtthermo`	`1000`	Frequency with which generates thermodynamical output.
`lammps_error`	`false`	Only restart those simulations for which an error occurred; valid options are `true` or `false`.
`lammps_momentum`	`100,1,1,1, angular`	Control zeroing of linear and angular momentum during execution; order of entries (separated by commas only!), i.e. `N,xflag,yflag,zflag,flag` (see LAMMPS manual); flag is omitted in the LAMMPS input script when the keyword `none` is used.
`lammps_nsample`	`1000`	Number of configurations used for averaging during execution of analysis routines.
`lammps_pdamp`	`1000`	Set the barostat damping constant for NPT simulations.
`lammps_prefix`	`1000`	Set project name as prefix to output files.
`lammps_tdamp`	`100`	Set the thermostat damping constant for both NVT and NPT simulations.
`lammps_tequil`	`1000`	Set the equilibration time
`lammps_tfreq`	`10`	Number of time steps skipped before adding a configuration to an average as sampled in input scripts.
`lammps_thermo_multi`	`false`	set thermo style to multi; valid options are `true` and `false`.
`lammps_triclinic`	`false`	Set triclinic mode; needed for simulation of triclinic boxes; valid options are `true` and `false`.
`lammps_trun`	`10000000`	Set run time; setting trun to `-` avoids its addition to job run scripts, thus not overriding different settings in subsequent chemistry files.

`location`	`analyze=.`, `field=.`, `include=.`	Prepend path for locations of analysis scripts, force field files, and include files; always included are for 1) `analyze`: directories `.` and `${EMC_ROOT}/scripts/analyze`, 2) `field`: directories `.` and `${EMC_ROOT}/field`, and 3) `include`: directories `.` and `${WORKDIR}/chemistry/include` when existing; multiple path entries are separated with a colon.
`mass`	`false`	Assume mass fractions in `chemistry.csv` input file.
`md_restart`	`false, ..`	Create restart scripts in chemistry file mode; this option allows setting an alternative data directory using in job run scripts, from which the most recent restart file will be used as a starting point; job run scripts will always try to restart – also when this option is set to `false` – from the specified data directory, creating a new serial directory by adding 1 to the current highest serial; serial numbers start at 00 and currently have a maximum of 99.
`md_shake`	`-`	Either switch off use of shake for select force fields or set masses, types, bonds, and/or angles for which to apply the SHAKE algorithm to; allowed keywords are `active`, `mass`, `type`, `bond`, `angle`; contributing types are separated by a colon ':', e.g. to freeze the angle of TraPPE water in LAMMPS use bond=hw:ow, angle=hw:ow:hw; see 'fix shake' in LAMMPS manual for shake interpretation; shake additions are only added when setting `active=true`.
`md_shake_iterations`	`20`	Set the maximum number of iterations used during SHAKE.
`md_shake_output`	`never`	Set the output frequency with which SHAKE statistics are written to the LAMMPS log file; options are integer numbers larger than 0 and `never`, which is equivalent to numbers smaller than 1.
`md_shake_tolerance`	`0.0001`	Set the SHAKE tolerance, which defines a successful SHAKE.
`md_shear`	`false`	Add shear paragraph to LAMMPS input script; valid options are `true` and `false`.

`md_timestep`	-	Set integration time step; default depends on force field.
`memorypercore`	`default`	Set memory per core in gigabyte for executed jobs; equivalent to `queue_memory`.
`mol`	`true`	Assume mol fractions in `chemistry.csv` input file.
`momentum`	`100,1,1,1, angular`	Depricated; control zeroing of linear and angular momentum during LAMMPS execution; order of entries (separated by commas only!), i.e. `N,xflag,yflag,zflag,flag` (see LAMMPS) manual; flag is omitted in the LAMMPS input script when the keyword `none` is used.
`moves_cluster`	`active=false, cut=0.05, frequency=1, limit=auto:auto, max=0:0, min=auto:auto`	Define cluster move settings used to optimize builds; multiple options are available, where option `active` switches the move on or off, `cut` sets a percentage of the maximum allowed displacement and rotation at which to turn the move off, `frequency` sets the frequency of the move, `limit` sets the maximum allowed displacement (in force field units) and rotation (in rad), `max` sets estimation for 50% acceptance of displacement and rotation (0:0 will trigger internal estimation), and `min` sets explicit rather than relative conditions to turn the move off.
`msd`	`false`	Add mean square displacement analysis to LAMMPS output; options are `true` or `false`.
`namd`	`false`	Create NAMD input script and parameter file.
`namd_dtcoulomb`	`1`	Set electrostatic interaction update frequency.
`namd_dtdcd`	`10000`	Set frequency with which snapshots are written to a DCD file.
`namd_dtnonbond`	`1`	Set nonbond interaction update frequency.
`namd_dtrestart`	`100000`	Set output frequency of restart files.
`namd_dtthermo`	`1000`	Set output frequency of thermodynamic quantities.
`namd_dttiming`	`10000`	Set timing frequency.

`namd_dtupdate`	`20`	Set update frequency.
`namd_pres_decay`	`50`	Set pressure ensemble decay.
`namd_pres_period`	`100`	Set pressure ensemble period.
`namd_temp_damp`	`3`	Set temperature ensemble damping.
`namd_tminimize`	`50000`	Set number of initial minimization steps.
`namd_trun`	`10000000`	Set total number of timesteps for execution run.
`niterations`	`1000`	Controls the number of iterations used for inserting an atom during the build process.
`norestart`	`false`	Control the capability for using run scripts – as resulting from environment scripts – for restarting; options are `true` or `false`; the latter means, that restarting with the same run script is possible.
`nparallel`	`auto`	Set the number of repeat units in the direction parallel to a surface defined in the chemistry file; values larger than 1 override the `auto` setting.
`nrelax`	`100`	Alter the number of relaxation cycles used during the EMC building process.
`nsample`	`1000`	Depricated; number of configurations used for averaging during execution of LAMMPS analysis routines.
`nthreads`	`1`	set number of cores for per thread for MD jobs.
`ntotal`	`10000`	Total number of created sites or atoms.
`number`	`false`	Interpret column 3 of `SHORTHAND` or `CLUSTERS` as the number of desired clusters; this number is an integer (see Section 4.6.2 [Chemistry File], page 38).
`omit`	`false`	Omit fractions from chemistry file; options are `true` or `false`.
`outer`	-	Set outer cutoff; note that `inner` and `outer` cutoffs are interpreted as fractions for colloidal force fields.

`pair`	`a=25, gamma=4.5, r=1`	Set DPD pair constants; only definite positive values are allowed.
`parameters`	`parameters`	Alternative parameter file name; the extension `.csv` is implied.
`params`	`true`	Create force field parameter file (`.prm`); currently only for DPD; options are `true` or `false`.
`pdamp`	`1000`	Depricated; set the LAMMPS barostat damping constant for NPT simulations.
`pdb`	`true`	Create PDB and PSF output files.
`pdb_atom`	`index`	Set atom name behavior in PDB and PSF output; options are `detect`, `index`, and `series`.
`pdb_compress`	`true`	Control compression of produced PDB and PSF files; options are `true` or `false`.
`pdb_connect`	`false`	Add connectivity to PDB; options are `true` or `false`.
`pdb_cut`	`false`	Cut bonds in PSF output, which span the simulation box; options are `true` or `false`.
`pdb_extend`	`false`	Use extended format in PSF output; options are `true` or `false`.
`pdb_fixed`	`true`	exclude flagged fixed sites when unwrapping clusters; options are `true` or `false`.
`pdb_hexadecimal`	`false`	Set hexadecimal output in PDB files; options are `true` or `false`.
`pdb_parameters`	`false`	Toggle output of NAMD parameter file; options are `true` or `false`.
`pdb_pbc`	`true`	Toggle application of periodic boundary conditions; options are `true` or `false`.
`pdb_rank`	`false`	Apply rank evaluation for coarse-grained output; options are `true` or `false`.

pdb_residue	index	Set residue name behavior in PDB and PSF output; options are detect, index, and series.
pdb_rigid	true	exclude flagged rigid sites when unwrapping clusters; options are true or false.
pdb_segment	index	Set segment name behavior in PDB and PSF output; options are detect, index, and series.
pdb_unwrap	clusters	Set PDB unwrapping mode; options are none, clusters, or sites.
pdb_vdw	true	Add Van der Waals representation to the VDW script created by EMC; options are true and false.
percolate	false	Treat imported structures as percolating InsightII structures; uses boundary crossing definitions as provided in the .mdf file; behavior of crystal option might be preferred; options are true or false.
phases	all	Sets which clusters to assign to each phase; each phase is separated by a +-sign; default assigns all clusters to phase 1; build order is defined by option build_order
polymer	bias=none, fraction=number, niterations=-1, order=list	Set group polymer distribution global defaults; bias helps in limiting construction time by controlling exclusion of unsuccessful subpolymers during polymer construction; valid bias options are none for no bias, binary for exclusion of subpolymers on a binary basis, and accumulative for exclusion on an accumulative basis; valid fraction options are number for using number of molecules and mass molecule mass as distribution entries; valid order options are list for sequentially and random of randomly interpreting the provided polymer distribution; the niterations keyword sets the maximum allowed number of iterations used during polymer sequence determination
polymer_niters	-1	Number of iterations for polymer construction.
port	-	Port EMC setup variables to other applications.
precision	0.001	Provide kspace long range Ewald summation precision.

`prefix`	`false`	Depricated; set project name as prefix to LAMMPS output files.
`pressure`	`false, direction= x+y+z, couple`	Set the system pressure and invoke an NPT ensemble; optionally specify barostat direction and/or directional coupling; valid directions are `x`, `y`, and `z`; valid coupling options are `couple`, `uncouple`, `true`, and `false`, where `true` corresponds to coupled directions; definition of the coupling direction follows the `couple` keyword through separation by a colon; a `+` sign separates multiple directions, e.g. atomistic `1,couple=y+z` sets a pressure of 1 atm in all directions with coupling in `y` and `z` directions; omission of coupling directions assumes an isotropic barostat.
`profile`	`density=false, density3d=false, pressure=false`	Generate density and/or pressure profiles of all clusters while executing LAMMPS; options are `density`, `density3d`, or `pressure`, followed by either `true` or `false`, separated by a colon; `true` is assumed when omitted; options are separated by a comma.
`project`	-	Set the project name; derived from setup script name when not given.
`quiet`	-	Supress all output generated by this setup script.
`radius`	`5`	Alter the radius of the spherical volume in which already built sites are relaxed; used during the EMC building process.
`record`	`cut=false, frequency=1, inactive=true, name="", pbc=true, unwrap=true`	Defines the record entry in the build paragraph, which records the build process by outputting a set of PDBs and PSFs; valid options are `cut` for cutting bonds in the last recorded frame, `frequency` for setting the frequency with which the relaxation process is written in each separate PDB, `inactive` for whether to add inactive entries to PDBs (denoted by positions (0,0,0)), `name` for the base name of the written PDBs and PSFs, `pbc` for mapping clusters back into the periodic box by their center of mass, and `unwrap` to toggle unwrapping of clusters.
`references`	`references`	Provide an alternative name for the references file; the extension `.csv` is implied.

`region_epsilon`	`0.1`	Set epsilon for excluded regions when importing structures; the given value is expected to be positive; other are ignored.
`region_sigma`	`1`	Set sigma for excluded regions when importing structures; the given value is expected to be positive; other are ignored.
`replace`	`false`	Replace or overwrite all written script files.
`restart`	`false,..`	Depricated: use `md_restart`; create LAMMPS restart scripts in chemistry file mode; this option allows setting an alternative data directory using in job run scripts, from which the most recent restart file will be used as a starting point; job run scripts will always try to restart – also when this option is set to `false` – from the specified data directory, creating a new serial directory by adding 1 to the current highest serial; serial numbers start at 00 and currently have a maximum of 99.
`rlength`	`-`	Provide a reference length; used when selecting non-dimensional force fields.
`rmass`	`-`	Provide a reference mass; used when selecting non-dimensional force fields.
`rmax`	`-1`	Set maximum build cutoff; only applicable to `field` `dpd` and `gauss`, for which defaults are `1` and `1.5` respectively.
`rtype`	`-`	Provide a reference type; used when selecting non-dimensional force fields.

`sample`	`energy=false, gyration=false, msd=false, pressure=true, volume=false, green-kubo=false`	Set sampling sections in LAMMPS input script for averaging of `energy` for itemized energetic contributions, `gyration` for radii of gyration distributions, `msd` for mean square displacements, `pressure` for the pressure tensor, `volume` for the volume tensor, or `green-kubo` for Green-Kubo output concerning correlations between the off-diagonal pressure tensor contributions as used in viscosity calculations; output files will have their respecitive identifier as extension; options are either `true` or `false`, expect for keyword `msd` which has `average` as additional option for time averaged mean square displacement.
`script`	`chemistry`	Set script default file name.
`script_ncolums`	`80`	Set number of colums in output scripts.
`seed`	`-1`	Provide initial random seed; a `-1` seed will invoke the use of the number of seconds since January 1, 1970 as an initial seed.
`shake`	-	Depricated: use `md_shake`; either switch off use of shake for select force fields or set masses, types, bonds, and/or angles for which to apply the SHAKE algorithm to; allowed keywords are `active`, `mass`, `type`, `bond`, `angle`; contributing types are separated by a colon ':', e.g. to freeze the angle of TraPPE water in LAMMPS use bond=hw:ow, angle=hw:ow:hw; see 'fix shake' in LAMMPS manual for shake interpretation; shake additions are only added when setting `active=true`..
`shake_ iterations`	`20`	Depricated: use `md_shake_iterations`; set the maximum number of iterations used during SHAKE.
`shake_output`	`never`	Depricated: use `md_shake_output`; set the output frequency with which SHAKE statistics are written to the LAMMPS log file; options are integer numbers larger than 0 and `never`, which is equivalent to numbers smaller than 1.
`shake_tolerance`	`0.0001`	Depricated: use `md_shake_tolerance`; set the SHAKE tolerance, which defines a successful SHAKE.

`shape`	`1`	Provide the desired shape of the simulation box; the shape factor refers to the relative fration of `lx/ly`.
`skin`	-	Depricated: use `lammps_skin`; set LAMMPS skin.
`shear`	`false`	Depricated: use `md_shear`; add shear paragraph to LAMMPS input script; options are `true` or `false`.
`split`	`phase=1, fraction=0.5, thickness=1, type=relative, mode=random, sites=all, groups=all, clusters=all`	Defines the selection criteria for applying a fractional split of clusters at the edges of the box; `phase` sets the phase to which to apply the split; `fraction` specifies the fractional split; `thickness` sets the thickness of the region taken into consideration; `type` sets the units of the thickness (valid options are `absolute` and `relative`); `mode` specifies the selections algorithm (valid options are `distance` and `random`); `sites` sets the sites to include in the selection (valid options are `all` for all available sites or a selected list of site types separated by colons); `groups` sets the groups to include in the selection (valid options are `all` for all available sites or a selected list of group ids separated by colons); `sites` sets the sites to include in the selection (valid options are `all` for all available clusters build in the set `phase` or a selected list of cluster ids separated by colons).
`suffix`	`_$HOSTNAME`	Set EMC and LAMMPS suffix.
`system`	`charge=true, geometry=true, map=true, pbc=true, id=main`	Perform various system checks or set system id; `charge` checks if the total system charge equals to zero; `geometry` checks if already exisiting clusters span newly created volume upon adding of a new phase; `map` allows for mapping box geometry to its minimum image; `pbc` applies periodic boundary conditions to existing molecules when changing box shape; valid options are `true` and `false`; `id` sets the id used for the targetted system.
`system_charge`	`true`	Check if the total system charge equals to zero; valid options are `true` and `false`.
`system_geometry`	`true`	Checks if already exisiting clusters span newly created volume upon adding of a new phase; valid options are `true` and `false`.

system_id	main	Set the EMC system ID; a string is expected.
system_map	true	Map the active system box to its minimum shape; valid options are true and false.
system_pbc	true	Apply periodic boundary conditions after building; valid options are true and false.
temperature	300 or 1	Provide the simulation temperature; atomistic and dimensionless force fields use 300 K and 1 respectively as default.
tequil	1000	Depricated: use lammps_tequil; set the LAMMPS equilibration time
tfreq	10	Depricated: use lammps_tfreq; number of time steps skipped before adding a configuration to an average as sampled in LAMMPS input scripts.
thermo_multi	false	Depricated: use lammps_thermo_multi; set LAMMPS thermo style to multi.
tighten	false	Set margin for tightening the simulation box when importing structures; options are false or a value to represent the desired margin, e.g. 3 Angstroms for atomistic systems; note, that tightening only occurs in the chosen build direction (see direction option).
timestep		Provides the LAMMPS MD time step.
timestep	-	Depricated: use md_timestep; set integration time step.
trace	false	Provide function trace upon execution error;
triclinic	false	Depricated; set LAMMPS triclinic mode; needed for simulation of triclinic boxes; valid options are true and false.
trun	10000000	Depricated: use lammps_trun; set LAMMPS run time; setting trun to - avoids its addition to job run scripts, thus not overriding different settings in subsequent chemistry files.

`types`	`false`	Output types only.
`units`	-	Set type of units; valid options are `reduced`, `real`, and `si`; alternatively, the outdated option `lj` can be used instead of `reduced`.
`units_energy`	-	Set units for energetic scale.
`units_length`	-	Set units for length scale.
`volume`	`false`	Assume volume fractions in `chemistry.csv` file; options are `true` or `false`.
`wall`	`10`	Set the thickness of a temporary wall for imposing walls between phases; valid options are `true`, `false`, or a positive number; `true` switches wall exclusion on without altering its value (see also option `exclude`).
`warn`	`true`	Control output of warning information; options are `true` or `false`.
`weight`	`bond=0.0001, focus=1, nonbond=0.0001`	User provided energetic weight for building procedure controlling nonbonded, bonded, and focussed interactions; the latter applies to imported structures.
`width`	`false`	Sets double width for generated output scripts; options are `true` for 160 characters or `false` for 80 characters.

4.6 File Formats

All entries within files are pasted into the resulting EMC scripts and force field files. Use of a chemistry file is mandatory. Use of references and parameters files is optional. All entries are assumed to be comma separated unless otherwise stated. All `ITEM` identifiers can be followed by an optional `comment` keyword, which can be set to either `true` or `false`.

```
ITEM  COMMAND [comment=[true|false]]
```

The full `ITEM` paragraph will be ignored until the next `ITEM END`, when keyword `comment` is set to `true`.

4.6.1 Environment File

The environment file functions as a wrap-around to chemistry files. It allows for setting up multiple simulations within one file, including looping over variables, thus enabling setting up series of simulations scanning parameter space. The environment format allows for defining parameter scans and simulation definition in one succinct file, thus furthering compact overviews. The keyword `ITEM` is required to preceede the identifiers listed in the following table. The identifiers are listed in the preferred order in which the are to appear in the environment script.

Identifier	Description
`ANALYSIS`	Controls the type of analysis performed during execution of analysis scripts as generated in directory `analyze`; this paragraph allows to include predefined or user-defined analysis scripts into a project's analysis; sample scripts can be found in `${EMC_ROOT}/scripts/analyze`; current supported types are `cavity`, `density`, `energy`, `green-kubo`, `last`, `pressure`, and `volume`; option `active` controls inclusion of the analysis type; valid values are `true` or `false`; options `archive`, `dir`, `skip`, and `window` are automatically passed to the analysis script, bearing internal settings; these options can be overridden, but is not advised; options and values are separated by an equal sign (`=`); different options are separated by a comma, tab or space; location of scripts is looked for in `${EMC_ROOT}/scripts/analyze/`, `chemistry/analyze`, or the path associated with the indicated script
`CLUSTERS`	Sets general cluster definitions which replace the `@{CLUSTERS}` reference in the chemistry template in the same `STAGE` and `TRIAL` section; `CLUSTERS` contents is stored in `chemistry/clusters/$stage/trial.dat`
`COMMENTS`	Sets the start of a comments section, which can only be terminated by a matching `END`
`END`	Marks the end of a paragraph

`ENVIRONMENT` Sets optional variables related to the environment (formerly `OPTIONS`); environment options are defined previously (see ⟨undefined⟩ [Environment Options], page ⟨undefined⟩)

`FIELD` Allows for a direct definition of force field parameters within the environment concept; the resulting `.prm` parameter file and `.top` topology file are stored in `chemistry/field/$stage/project.{prm|top}` and should be refered to by using this location (see Section 4.6.3 [Field File], page 48)

`GROUPS` Sets general group definitions which replace the `@{GROUPS}` reference in the chemistry template in the same `STAGE` and `TRIAL` section; `GROUPS` contents is stored in `chemistry/groups/$stage/trial.dat`

`INCLUDE` Specifies a files to be included containing a subset of commands; the file name follows `INCLUDE` directly; `INCLUDE` is a single line item and is not closed with an `END`

`LOOPS` Lists variables over which to loop; variables are expected to be lower case; variables `stage`, `trial` and `copy` are reserved; variables can be paired or coupled by adding `:p` behind the variable, which couples the current variable to the previous; add `:h` in order to hide or exclude the variable from the data directory structure; an `:h` automatically implies the variable is paired with its predecessor; add `:d` to the predecessor when double occurences are intended (useful in case of pairing); alternatively, permutations in variables can be accomplished by adding `:2` for pairs, `:3` for triplets, and `:4` for quadruplets; focus on on component of the produced list can be obtained by adding a second colon followed by a number, e.g. `name:2:1` would create all permutations for pairs which include element 1 for variable `name`; element counting starts at 0; later reference to any and all variables recorded in `LOOPS` are to be preceeded with `@`, enclosed within curly brackets `{}`, and written in all-caps, e.g. `@{NAME}` refers to variable `name`; these references can also be used in subsequent loop variables

`PARAMETERS` Allows for direct inclusion of DPD parameters; the resulting parameter file is stored under `chemistry/field/$stage/parameters.csv` (see Section 4.6.5 [Parameters File], page 53)

`POLYMERS` Sets general polymer definitions which replace the `@{POLYMERS}` reference in the chemistry template in the same `STAGE` and `TRIAL` section; `POLYMERS` contents is stored in `chemistry/polymers/$stage/trial.dat`

`REFERENCES` Allows for direct inclusion of DPD references; the resulting reference file is stored under `chemistry/field/$stage/references.csv` (see Section 4.6.4 [References File], page 52)

`SHORTHAND` Sets general shorthand definitions which replace the `@{SHORTHAND}` reference in the chemistry template in the same `STAGE` and `TRIAL` section; `CLUSTERS` contents is stored in `chemistry/shorthand/$stage/trial.dat`

`STRUCTURES` Defines a list of structures, relating directly to `trial` loop entries; structures are referred to by `@{STRUCTURE}`

`STAGE` Indicates the start of a section relating to loop variable `stage`; `STAGE` is followed by a stage indicator, which should appear in the enumerated loop variable `stage`; all following paragraphs relate to the indicated stage; `STAGE` is a single line item and is not closed with an `END`; all identifiers following `STAGE` refer to loop variable `stage`

`TEMPLATE` Defines a chemistry template; templates are stored in `chemistry/stages/stage.esh`; templates are used to create complete chemistry files upon execution of the associated run script; references to loop variables start with `@{}` and are written in all caps; references are treated as environment variables

`TRIAL` Indicates the start of a section relating to loop variable `trial`; `TRIAL` is a single line item and is not closed with an `END`; all identifiers following `TRIAL` refer to loop variable `trial`

`VARIABLES` Lists extra enviroment variables; environment variables are referred to with an at symbol `@` and are enclosed by curly brackets `{}`; reserved variables are `@{EMCROOT}` for the root location of EMC and `@{WORKDIR}` for the directory in which the collection of simulations takes place

`WRITE` Specify a message line to be written to the output; the written text directly follows `WRITE`; `WRITE` is a single line item and is not closed with an `END`

EMC setup only assumes environment modus, when the `environment` variable is set to `true` in the first occurring `OPTIONS` paragraph. Enviroment mode allows for optionally setting names associated with build, analyze, and run scripts, as well as defining which queues to be used. Dashes imply undefining parameters. The `ncores` parameter is mandatory. For certain problems, it can be desired to simulate multiple indepedent structures in order to obtain a correct statistical sample. To this end, a loop variable `copy` is available, which creates multiple copies with the same conditions, but executing building with different random seeds, thus creating indepedent initial structures. Note, however, that use of this features quickly creates many simulations, which harbors the danger of overcrowding high-performance computing queueing systems and over-requesting available computational resources. An example for environment mode application can be found in `${EMC_ROOT}/examples/setup/environment/shear`.

4.6.2 Chemistry File

The chemistry file – with default file name `chemistry.esh` – supports environment, new-style, and legacy formats. Columns can only be separated by commas or tabs. Next line extension occurs by using an ampersand `&` at the end of to be extended line.

4.6.2.1 General

The new-style format allows for the definition of polymers, which uses keyword `ITEM` followed by an identifier to distinguish between different contributing paragraphs, as described by

Identifier	Description
`CLUSTERS`	Sets cluster definitions, using previously defined cluster groups
`COMMENTS`	Sets the start of a comments section, which can only be terminated by a matching `END`
`EMC`	Defines a section to be copied verbatim into the EMC build script; optionally, the keyword can be followed by either one or two numeric values, identifying which phase and where the block verbatim text is to be added to; alternatively, identifiers `phase` and `spot` can be used, followed by an equal sign (`=`); valid values are the respective phase for identifier `phase` and a number between 0 and 2 for `spot` to place the defined section at specific spots in the selected phase section in the EMC build script; defaults for phase and spot are 1 and 0 respectively when value and identifiers are omitted;
`END`	Marks the end of a paragraph
`FIELD`	Allows for a direct definition of force field parameters; the resulting `.prm` parameter file and `.top` topology file are stored in the current directory (see Section 4.6.3 [Field File], page 48)
`GROUPS`	Sets separate group definitions, including polymeric repeat units
`INCLUDE`	Specifies a files to be included containing a subset of commands; the file name follows `INCLUDE` directly; `INCLUDE` is a single line item and is not closed with an `END`

`LAMMPS` Defines a section to be copied verbatim into the LAMMPS execution script, appearing atop the simulation section; alternatively, identifiers `stage` and `spot` can be used, followed by an equal sign (`=`); allowed options for `stage` are – in order of appearance in the LAMMPS execution script – `header`, `variables`, `interaction`, `equilibration`, `simulation`, `integrator`, `sampling`, `intermediate`, `run`; allowed options for `spot` are `head` and `tail`; stage options correspond to the comments in the resulting LAMMPS execution script; defined additional variables as described above can be used when captured between `${..}`.

`OPTIONS` Allows for setting all command line options of `emc_setup.pl` inside chemistry files, thus creating the option of consolidating a simulation setup within one file only; see the previous paragraph; syntax is *option*,*value*[,...], where *option* refers to any of the options as defined previously (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩)

`POLYMERS` Defines polymers, using previously defined polymeric groups; this paragraph can be used for defining polymers either through the `GROUPS` or `CLUSTERS` paragraph

`PARAMETERS` Allows for direct inclusion of DPD parameters; parameters are not stored but used internally (see Section 4.6.5 [Parameters File], page 53)

`PROFILES` Defines additional mass profiles, where each line generates an additional profile; syntax is *name*, *mode*[:*type*[:*binsize*]], *contributor*[,...]; *mode* can be either `cluster` or `type`; *type* sets the type of profile, which can either be `density`, `density3d`, or `pressure`; `density` is assumed when omitted; *binsize* sets the individual bin size; the general binsize is assumed when omitted; *contributor* refers to either a cluster name or type repectively; note, that profile coordinates are in reduced units and run from 0 to 1

`REFERENCES` Allows for direct inclusion of DPD references; references are not stored but used internally (see Section 4.6.4 [References File], page 52)

`SHORTHAND` Legacy format, allowing for group and cluster definitions in shorthand notation; polymeric definitions are not possible within this format

`VARIABLES` Defines additional variables, which are either used by read EMC structures from the EMC structure library ($EMC_ROOT/lib), serve as pass-through to subsequent simulation packages (e.g. LAMMPS), or are used in the generated EMC build script; the section identifier `VARIABLES` can be followed by keywords `head` for appearance at the top or `tail` for appearance at the bottom of the EMC script variable block

`WRITE` Specify a message line to be written to the output; the written text directly follows `WRITE`; `WRITE` is a single line item and is not closed with an `END`

4.6.2.2 Shorthand

The `SHORTHAND` format can be used without a line starting with keyword `ITEM`, provided its lines are at the start of the chemistry file. Conversely, the shorthand format also provides quick definition of non-polymeric molecules without the need of defining groups and clusters. Its format is given by the following table.

Entry	Description
1	Cluster and Sets the group name, which optionally can be followed by `f[ield]=id[,...]`, `m[ass]=#`, or `t[erm]` – separated by colons – where characters between `[]` are optional and can be omitted; multiple options can be set in one line; `f[ield]=id` sets a specific field `id` when using multiple fields for typing, e.g for CHARMM (see Section 5.11 [Field], page 106), where `id` can be part of the full identifier; `m[ass]=#` overrides automated setting of group mass, where `#` represents the desired mass; `t[erm]` marks group as a terminator (see Section 5.17 [Groups], page 117)
2	Chemical representation in the form of a SMILES string (see Section 7.8 [SMILES], page 197)
3	Fraction or count; can be either molecular, mass, or volume fraction, or the number of clusters based provided option (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩, for options `mass`, `mol`, or `number`)
4	Molecular mass in g/mol
5	Molecular volume in cc/mol

Column 4 and 5 are optional.

4.6.2.3 Groups

The `GROUPS` paragraph defines multiple groups, including polymeric groups. Its format is given by the following table.

Entry	Description
1	Sets the group name, which optionally can be followed by `f[ield]=id[,...]`, `m[ass]=#`, or `t[erm]` – separated by colons – where characters between `[]` are optional and can be omitted; multiple options can be set in one line; `f[ield]=id` sets a specific field `id` when using multiple fields for typing, e.g for CHARMM (see Section 5.11 [Field], page 106), where `id` can be part of the full identifier; `m[ass]=#` overrides automated setting of group mass, where `#` represents the desired mass; `t[erm]` marks group as a terminator (see Section 5.17 [Groups], page 117); note, that groups with only one connection are automatically terminators

2 Sets the group chemistry using SMILES strings (see Section 7.8 [SMILES], page 197) or polymer type to turn on polymer mode; in SMILES polymeric connectivity points are marked with `*`; valid polymer types are `alternate`, `block`, and `random`; column 3 and onwards are omitted when using groups in polymer mode; the polymer distribution is defined through the `POLYMERS` paragraph

3 Defines connection index for polymeric groups of current group; indices refer to `*` occurences in the previously set SMILES string

4 Defines the connecting group `name` and corresponding `index` in the format `name:index`

5- Repeats of column 3-4 for subsequent connections

4.6.2.4 Clusters

The `CLUSTERS` paragraph defines multiple clusters. Options following keyword `import` in entry 2 can be used in any order when superceeded by their respective identifier and separated by an `=` sign. Column 4 and onward are optional and have different meaning depending on the mode in which the cluster option is called. In case of polymeric definitions, column 4 through 5 are used. In case of imported structures, all columns 4 and onward can be used. Identifiers instead of columns can be used when importing structures. Each identifier is described in the paragraph and table hereafter. The use of commas or tab as option separators between entries is advised when defining an import. Its format is given by the following table, where the identifier column refers to the next table.

Entry	Identifier	Description
1	-	Sets the cluster name
2	-	Sets the associated group, the type of polymer – indicated by `alternate`, `block`, or `random` – or whether to expect a imported structure from file as incidated by keyword `import`; consequently, group names are not allowed to have reserved names `alternate`, `block`, `random`, or `import`
3	- or `ncells`	Sets a fraction or count, which can be either a mol, mass, or volume fraction, or number of clusters, based provided command line option (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩, for options `mass`, `mol`, or `number`); in case of keyword `import` in entry 2, this entry contains the number of repeat units (id `ncells`)
4	- or `name`	Molecular mass in g/mol or file name (id `name`) in case of keyword `import` in entry 2

5	- or `mode`	Molecular volume in cc/mol or import mode (id `mode`) when selecting keyword `import` in entry 2
6	`type`	Optional entry for import type
7	`flag`	Optional entry for setting mobility flags of imported structures
8	`density`	Optional entry for setting the density treatment of imported structures
9	`focus`	Optional entry for marking the imported structure as focus
10	`tighten`	Optional entry to trim or tighten imported structures
11	`ntrial`	Optional entry for setting the number of trials
12	`periodic`	Optional entry for setting periodicity; currently not implemented
13	`field`	Optional entry for enforcing a specific force field for force field typing of the imported structure;
14	`exclude`	Optional entry for setting an exclusion region
15	`depth`	Optional entry for setting ring recognition recursive depth
16	`percolate`	Optional entry for identifying a percolating crystalline structure
17	`unwrap`	Optional entry for unwrapping imported structures
18	`guess`	Optional entry for guessing unwrap status based on input
19	`charges`	Optional entry for using charges from imported morphologies
20	`formal`	Optional entry for using formal charges from imported morphologies when available
21	`translate`	Optional entry for translating the all imported sites for a distance into the growth direction of the simulation box
22	`map`	Optional entry for mapping both sites and box shape unto their periodic minimum image

The following table describes import identifiers, which can be used without having to abide by strict column sequences. Identifiers and their values are separated by equal signs (`=`), omitting any spaces. Examples are mentioned after this table.

Identifier	Default	Description
charges	false	Use charges from imported structures; allowed values ar true or false
density	mass	Sets density treatment of imported structures, which is needed for calculating correct box dimensions when adding material; allowed values are mass or number
depth	auto	Sets the maximum size of rings to be recognized; the provided input should be a definite positive integer or keyword auto
exclude	true	Toggles whether to add an repulsive exclusion region around the imported structure; allowed values ar true or false
field	-	Enforces a specific force field for force field typing of the imported structure
flag	rigid	Sets mobility flag for imported structures; allowed flags are fixed, rigid, or mobile
focus	true	Toggles whether to focus on the imported structure, which excludes its volume for additionally built molecules; allowed values are true or false
formal	true	Include formal charges from imported morphologies when provided in input structure; allowed values are true or false
guess	auto	Sets optional guessing of unwrapping based on imported structures; only functions when unwrap is set to auto; allowed values are auto, true, or false
map	false	Map both sites and box geometry onto their periodic minimum image; allowed values are true or false
mode	emc	Optionally enforce import mode when not using name extensions in identifier name; allowed modes are emc for EMC files, pdb for PDB files – providing both .pdb and .psf files, or insight for Insight II files – providing both .car and .mdf files
name	-	Sets file name; allowed file name extensions are .emc for EMC files, .car or .mdf for InsightII files, and .pdb or .psf for PDB files; default is .emc; extensions override mode defaults

`ncells`	`1:auto:auto`	Sets the number of repeated unit cells along the direction as set by option `direction` in the options paragraph (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩); repeats in multiple directions can be set when dividing input by colons in order of x, y, and z direction respectively; default is `1` in the main direction and `auto` in the remaining directions, e.g. `auto:1:auto` for main direction y
`ntrials`	`10000`	Sets the number of trial iterations used for determining the volume of an import of `type=line` or `structure` (see column 6), which employs a Monte Carlo algorithm; option is a number larger than zero
`percolate`	`auto`	Sets whether imported structure reflects a percolating crystal; allowed values are `auto`, `true`, or `false`
`periodic`	-	Sets periodicity; currently not implemented
`tighten`	-	Set the tightening distance, which effectively shrink-wraps the box around what is imported; allowed option is a distance; default is a distance of `1` or `3` for coarse grained or atomistic force fields respectively
`translate`	-	Translate by a length all sites into the direction of box growth; the entered value represents a scalar, which internally is multiplied by the unit box vector in the direction of growth; by default no translation is performed
`type`	`surface`	Sets the imported structure type; allowed types are `crystal` when adding no additional sites, `surface` when adding sites in one direction as set by option `direction` in the options paragraph (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩), `tube` when adding in two directions, `structure` when adding in three directions, or `system` when importing a previously built system
`unwrap`	`false`	Sets whether to unwrap the imported structure; allowed values are `auto`, `true`, or `false`

Columns 4 is optional when defining molecules, but mandatory when importing structures. Importing options defined in columns 3 through 14 can be preceeded by their respective identifiers, i.e. `ncells`, `name`, `mode`, `type`, `flag`, `density`, `focus`, `tighten`, and `ntrials`, separated by an equal sign. The order as given in the table above is assumed when omitting this identifier. The specified default is taken, when the an option is omitted. A formal example of an import line is given by

```
surface     import, name=$root+"lib/fcc", ncells=8, mode=emc, &
```

```
type=surface
```

which creates a surface with cluster name `surface`, using a nonbonded fcc lattice from the EMC library, consisting of 8 repeat units in the x-direction. Note, that any given order can be used, when identifiers are specified. A shorter, but equivalent definition is given by

```
surface     import  8       $root+"lib/fcc" emc
```

where now the option order matters, therefore creating the need for strictly following the column order as given in the above table. Alternatively, the import mode can be determined by EMC setup through adding the file extension, i.e.

```
surface     import name=$root+"lib/fcc.emc" ncells=8 type=surface
```

or

```
surface     import  8       $root+"lib/fcc.emc"
```

Note, that a space or comma can also be used as separator. However, this option is not advised when using a space or comma in the file name. Currently, only one structure at a time can be imported by EMC.

4.6.2.5 Polymers

In the case of a polymer, the cluster name has to appear in the `POLYMERS` paragraph, when defining the type of polymer in column 2. Otherwise, a group name is assumed. The `POLYMERS` paragraph describes the composition of one polymer at a time. Each polymer definition starts with the polymer's cluster name on a separate line. The polymer name optionally can be followed by keywords `fraction`, `niterations`, or `order`, separated from their value by an `=` sign. Defining any of these keywords overrides global definitions for the indicated polymer only (see keyword `polymer` under `OPTIONS` for globals (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩)). Valid `fraction` options are `number` for using number of molecules and `mass` molecule mass as distribution entries. Valid `order` options are `list` for sequentially and `random` of randomly interpreting the provided polymer distribution. The `niterations` keyword sets the maximum allowed number of iterations used during polymer sequence determination. Lines following the polymer name define the polymer's distribution of contributing groups. Multiple polymer definitions can be privided within this paragraph. The format of contributing groups is given by the following table.

Entry	Description
1	Mol or mass fraction of the polymer cluster defined by this line; mass fractions are only allowed when defining polymers through groups

2 Group name; multiple group names can be given through separation with a ':'; weights – with which groups are randomly picked – follow an '=' and are separated by a ':', e.g. `A:B=1:2`; equal weights are assumed when omitted

3 Number of repeat units for the above group

4- Repeats of columns 2-3 for subsequent groups

Capping or end groups should be included in the above list for mass calculation purposes, although they are neglected by EMC. EMC selects end groups internally.

4.6.2.6 DPD Additions

The following identifiers are valid only, when selecting DPD as force field:

Identifier	Description
`ANGLES`	Defines a paragraph for additional specific angle entries, which should be provided in the order of `type1`, `type2`, `type3`, `k`, `theta`
`BONDS`	Defines a paragraph for additional specific bond entries, which should be provided in the order of `type1`, `type2`, `k`, `l`
`MASSES`	Defines or – in case of replicas – redefines masses, which should be provided in the order of `type`, `mass`; `mass` is a definite positive number and has reduced units when no reference type has been chosen; otherwise, `mass` is in units of the reference type (see Section 4.4 [Options], page 12)
`NONBONDS`	Defines a paragraph for nonbond entries, which should be provided in the order of `type1`, `type2`, `a`[, `cutoff`[, `gamma`]]; these entries overwrite already exisiting entries
`REPLICAS`	Creates duplicates of exisiting nonbond entries, which should be provided in the order of `target`[:`factor`], `source`[:`fraction`[:`flag`]][, `source`[:`fraction`[:`flag`]][, ...]], `offset`; interaction constants, mass, cutoff, and gamma are averaged when multiple sources are provided; `offset` defines the offset of the new DPD interaction constant a_{ij} with respect to the original; `factor` defines a multiplication factor with respect to the `offset`; `factor` is set to `1` when omitted; `fraction` defines which fraction to use of each contributing component; equal fractions are assumed when omitted; normalization of `fraction` is controlled by `flag`, which can either be `true` or `false`; by default `fraction` is normalized; `fraction` is not normalized when any one `flag` is set to `false`
`TORSIONS`	Defines a paragraph for additional specific torsion entries, which should be provided in the order of `type1`, `type2`, `type3`, `type4`, `k`, `n`, `delta`, and optionally following triplets of `k`, `n`, and `delta`

Both keyword and identifier must be in all caps and separated by a comma or tab. Currently, alterations or additions to force field files only function for the DPD force field. Types can be replicated in order to either create duplicates or compounded types, where

$$a_{ij}^{*} = f(a_{ij}^{*} - 25) + \Delta$$

expresses the new interaction parameter a_{ij}^{*} . Assume, that a new type `AIR` is to be created based on an alkane representation `C4`, where multiplication factor $f = 1.5$ and offset $\Delta = 25$. This would result in the following `REPLICAS` entry

```
AIR:1.5     C4      25
```

For each of the parameter additions (i.e. nonbonds, bonds, angles, or torsions), entries containing a wildcard `*` will appear in the `AUTO` paragraph of the resulting force field file.

4.6.3 Field File

The field file – using the extension `.define` – allows for defining force field parameters and rules within one file. Translation of this file into an EMC `.prm` parameter file and `.top` topology file occurs by applying `field.pl`, which can be found in the EMC script directory `${EMC_ROOT}/scripts`. This format can also be included in environment and chemistry files by adding the field definition between identifiers `FIELD` and `END`, both preceeded by keyword `ITEM`.

4.6.3.1 General

The field format itself also uses keyword `ITEM` to identify subsequent main functionality paragraphs, as decribed by

Identifier	Description
`DEFINE`	Defines different force field settings as described below; this paragraph is mandatory
`MASSES`	Defines the types and their associated masses and subsequent definitions; the syntax is *type*, *mass*, *element*, *number of connections*, *formal charge*, *comment*; this paragraph is mandatory
`COMMENTS`	Allows for inclusion of comments specific to the force field's derivation; this paragraph is optional
`REFERENCES`	Reports the literature references used to construct the force field; the syntax is *year*, *volume*, *page*, *journal*; this paragraph is optional
`PRECEDENCE`	Defines the precedence table as described below; this paragraph is optional
`EQUIVALENCE`	Describes the equivalences of the force field; equivalences state which type can be used instead of the official type for abstracting the asociated force field parameters; the syntax is *type*, *pair*, *bond*, *angle*, *torsion*, *improper*; this paragraph is optional
`NONBOND`	Defines the nonbond parameters of the force field; the syntax starts with *type1*, *type2*, followed by the actual parameters (e.g. *epsilon*, *sigma* for Lennard-Jones force fields, and *a*, *cutoff*, *gamme* for a DPD force field); the expected number of parameters is dependent on the chosen force field mode; this paragraph is mandatory
`BOND`	Defines the parameters for bond length interactions; the syntax is *type1*, *type2*, *k*, *l0*; the interaction function depends on the the force field mode, but is in general harmonic in nature; this paragraph is optional

`ANGLE` Defines the parameters for bond angle interactions; the syntax is *type1*, *type2*, *type3* *k*, *theta0*; the interaction function depends on the the force field mode, but is in general harmonic in nature; this paragraph is optional

`TORSION` Defines the parameters for dihedral interactions; the syntax is *type1*, *type2*, *type3*, *type4*, *k*, *n*, *delta*[, *k*, *n*, *delta* ...]; the allowed number of additional parameter entries depends on the chosen force field mode; the interaction function depends on the the force field mode, but is in general a Fourier series of cosines (e.g. see Section 7.12.19 [Standard], page 341, Torsion paragraph); this paragraph is optional

`IMPROPER` Defines improper interactions; the syntax is *type1*, *type2*, *type3*, *type4*, *k*, *psi0*; the interaction function depends on the the force field mode, but is in general harmonic in nature; this paragraph is optional

`RULES` Defines the rules associated with the types as defined by the `MASSES` paragraph; the syntax is *type*, *partial charge*, *rule*[, *rule* ...]; one type can have multiple rules; a rule describes a unique chemical environment, which defines the type at hand; this paragraph is optional

`TEMPLATES` Defines templates, which can be used as aliases when defining group chemistry; this paragraph is optional

Each entry can be separated by either a `TAB`, `COMMA`, or `SPACE`. A `TAB` represents the preferred separator. Wildcards as represented by `*` can be used for types in parameter definitions. Partial wildcards are also allowed. The partial wild cards can be understood by the following example: assume `c4` generally describes an SP3 carbon with no hydrogen attached, `c4h` with one hydrogen attached, and `c4h2` with two hydrogens attached, then wildcard `c4*` would describe all three of these types.

4.6.3.2 Define

Specific alternative settings can be defined for force field by using the `DEFINE` paragraph, as decribed by the following

Identifier	Description
`ANGLE`	Sets the error handling for typing of angle interactions; options are `ignore`, `complete`, `warn`, `empty`, and `error`
`CREATED`	Sets the creation date of the force field
`CUTOFF`	Sets the cut off for all nonbonded interactions

`DENSITY` Sets the unit of length as used by the parameters in the force field; options are `g/cc`, `kg/m^3`, and `reduced`; both `LENGTH` and `ENERGY` are set to `reduced` when the latter option is chosen, thus resulting in a force field in reduced units

`ENERGY` Sets the unit of length as used by the parameters in the force field; options are `j/mol`, `kj/mol`, `cal/mol`, `kcal/mol`, `kelvin`, and `reduced`; both `LENGTH` and `DENSITY` are set to `reduced` when the latter option is chosen, thus resulting in a force field in reduced units

`FFDEPTH` Sets the maximum recursive with which to trace the chemical surrounding of a site during typing; this depth should equal the maximum depth of the provided rules

`FFMODE` Sets the name or mode of the force field; options are `born`, `charmm`, `dpd`, `martini`, `mie`, `opls`, `sdk`, `standard`, and `trappe`

`FFTYPE` Sets the force field type; options are `atomistic`, `united`, and `coarse`

`IMPROP` Sets the error handling for typing of improper interactions; options are `ignore`, `complete`, `warn`, `empty`, and `error`

`INNER` Sets the inner cut off when using a force field with a switching function for nonboned interactions (e.g. as in the CHARMM force field (see Section 7.12.7 [CHARMM], page 236))

`LENGTH` Sets the unit of length as used by the parameters in the force field; options are `angstrom`, `nanometer`, `micrometer`, `meter`, and `reduced`; both `DENSITY` and `ENERGY` are set to `reduced` when the latter option is chosen, thus resulting in a force field in reduced units

`MIX` Sets the mixing rule; options are `none`, `bethelot`, `arithmetic`, `geometric`, and `sixth_power`

`NBONDED` Sets the number of bonded atoms to exclude from nonbonded interactions; the number varies by force field

`PAIR14` Sets the error handling for typing of 1-4 pair interactions; options are `off`, `false`, `exclude`, `on`, `true`, and `include`

`TORSION` Sets the error handling for typing of dihedral interactions; options are `ignore`, `complete`, `warn`, `empty`, and `error`

`VERSION` Sets the force field version

Error handling, as mentioned in the latter options of the above table, can be dealt with in a number of ways. Any errors can be either fully ignored without any output by EMC during typing by chosing option `ignore`. A warning will be generated, when chosing option `warn`. An empty parameter entry will be generated without any output to screen when chosing option `empty`. EMC will continue after all of these options. However, EMC will generate warnings for all missing parameters and will cease execution when chosing options `error`. Examples of `.define` files can be found for several force fields as provided in the `${EMC_ROOT}/field/` force field directory (e.g. the CHARMM, OPLS, and TraPPE force fields).

4.6.4 References File

Reference files are optional and are used to create comprehensive EMC force field files (using `.prm` extensions). Currently only DPD force fields are used when interpreting the reference file. Its default file name is `references.csv`.

Column	Description
1	Short ID as referred to in `parameters.csv`
2	ID as represented in produced `.prm` force field files
3	Bead mass in g/mol
4	Bead volume in nm^3
5	Number of connections that this bead can have
6	Effective charge of bead
7	Number of repeat units represented in this bead (e.g. 3 when 3 ethyleneoxide monomers are captured by one bead)
8	Comment describing the origin of the bead

Future `emc_setup.pl` versions are intended to span a wider choice of force fields when using reference files.

4.6.5 Parameters File

Parameter files are optional and are used to provide parameters. Currently they are only used in case of the DPD force field (see Section 7.12.11 [DPD], page 275). Its default file name is `parameters.csv`.

Column	Line	Description
1	1	Mandatory empty field
	2-	Alphanumeric type name of bead A (short)
2	1	Mandatory empty field
	2-	Alphanumeric type name of bead B (short)
3	1	Temperature at which the interaction parameters in this column were determined.
	2-	Numerical value refering to the interaction of bead A and B
...		Identical to column 3.

Column 4 and subsequent columns can hold parameters determined at different temperatures. Their syntax is the same as for entries in column 3. Future `emc_setup.pl` versions are intended to include a wider choice of force field when using parameter files.

4.7 Examples

The directory `${EMC_ROOT}/examples/setup/` holds a number of examples with different complexity, all using `emc_setup.pl` as a base for creating EMC build and LAMMPS input scripts. The examples are subdivided in chemistry and environment related examples. This section describes ways to build bulk systems with a mixture of chemicals, typing of various force fields, construction of various types of polymers, building multiphase simulations, and ways to build material between two surfaces. A second set of examples in `${EMC_ROOT}/examples/build/` illustrate adaptation of EMC scripts directly for specific solutions. These scripts are not discussed here.

4.7.1 References

1. VMD - Visual Molecular Dynamics, '`http://www.ks.uiuc.edu/Research/vmd/`'
2. Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", *J. Molec. Graphics* **1996**, *14*, 33-38
3. LAMMPS - Molecular Dynamics Simulator, '`http://lammps.sandia.gov/`'
4. Plimpton, S., "Fast Parallel Algorithms for Short-Range Molecular Dynamics", *J. Comput. Phys.* **1995**, *117*, 1-19

4.7.2 Chemistry Mode

Chemistry mode examples show how to set up single system simulations using a number of different common problems, which include setting up bulk systems, systems with different force fields, polymer systems, systems with multiple phases, and systems with surfaces. Chemistry examples can be found in `${EMC_ROOT}/examples/setup/chemistry/`.

4.7.2.1 Bulk Mixture

Assume the desire exists to simulate a system consisting of 45% water, 40% alcohol, 5% salt, and 10% sugar by mass, where the force field of choice is PCFF and consists of about 2000 particles once built. The setup utility offers multiple ways for setting up a simulation. The quickest way is to use emc_setup.pl's legacy format. For this format, one first creates a `chemistry.esh` file with the following content:

```
water       O,45
alcohol     CCO,40
salt        [Na+].[Cl-],5
sugar       OCC1OC(O)C(O)C(O)C1O,10
```

Note, that the fraction indications – 45, 40, 5, and 10 – in the last column are normalized internally, and can therefore be any number. Here, percentages are taken for illustrating purposes. The next step is to invoke `emc_setup.pl`,

```
emc_setup.pl -field=pcff -ntotal=2000 -mass -build_dir=. bulk
```

where `-field` selects the force field, `-ntotal` sets the number of particles, `-mass` assumes mass fractions in the `chemistry.esh` file, and `-build_dir` sets the origin of the created files to the current directory. An alternative chemistry file with the name `bulk.esh` uses shorthand notation in combination with options. Here all command line options are folded into the chemistry file, which reads as follows,

```
#!/usr/bin/env emc_setup.pl

# Options

ITEM        OPTIONS

replace     true
mass        true
ntotal      2000
field       pcff
density     1
build_dir   .

ITEM        END

# Shorthand

ITEM        SHORTHAND

water       O,45
alcohol     CCO,40
salt        [Na+].[Cl-],5
```

```
sugar       OCC1OC(O)C(O)C(O)C1O,10

ITEM        END
```

A more general way of describing the same simulations is by the use of `GROUPS` and `CLUSTERS`. EMC internally uses sites, groups, and clusters to represent atoms, repeat units, and molecules respectively. Using groups in case of small molecules makes less sense than using groups for polymers. However, as a general notation, this problem can also be defined by the following

```
#!/usr/bin/env emc_setup.pl

# Options

ITEM        OPTIONS

replace     true
mass        true
ntotal      2000
field       pcff
density     1
build_dir   .

ITEM        END

# Groups

ITEM        GROUPS

water       O
alcohol     CCO
salt        [Na+].[Cl-]
glucose     OCC1OC(O)C(O)C(O)C1O

ITEM        END

# Clusters

ITEM        CLUSTERS

water       water,45
alcohol     alcohol,40
salt        salt,5
sugar       glucose,10

ITEM                END
```

Execution can be performed in one of two ways,

```
emc_setup.pl bulk
```

or, in the case of a Unix-like operating system,

```
./bulk.esh
```

provided `bulk.esh` is executable and `emc_setup.pl` can be found in the shell's predefined path. This solution allows for a comprehensive representation of a simulation, without having a number of separate files describing it. Once executed, `emc_setup.pl` creates the EMC build script `build.emc` and the LAMMPS input script `bulk.in`. To build an input structure, the EMC build script needs to be executed using EMC. Assume, that the LINUX version `emc_linux` is used, which location is in your path variable `$PATH`. Then,

```
emc_linux build.emc
```

will create input structures in `PDB` format – `bulk.pdb` and `bulk.psf` – and `LAMMPS` format – `bulk.in`. The resulting structure can be inspected by means of VMD (Visual Molecular Dynamics)[1,2] through

```
vmd -e bulk.vmd
```

assuming, that `vmd` is in your path. The advantage of using `bulk.vmd`, is that this file contains the correct Van der Waals radii for the simulated beads. A molecular dynamics simulation with LAMMPS (Large-scale Atomic/Molecular Massively Parallel Simulator)[3,4] can be started with

```
lmp_linux < bulk.in
```

assuming, that `lmp_linux` is in your path. A similar example can be found in the `${EMC_ROOT}/examples/setup/chemistry/bulk` directory.

4.7.2.2 Force Fields

EMC allows for the use of various force fields. To illustrate this, five examples are included for commonly used force fields, which are separated into atomistic (CHARMM, OPLS-AA, PCFF), united atom (OPLS-UA, TrAPPE), and coarse grained (DPD) examples. Force field choice is indicated by the `-field` flag of the `emc_setup.pl` script, which will try to find a corresponding force field in the `${EMC_ROOT}/field` directory. For the latter to work, it is assumed, that `emc_setup.pl` is located in the `${EMC_ROOT}/scripts` directory. With exception of the coarse-grained example, all examples use a molar ratio of 1:4 for a ethanol:water mixture. The corresponding chemistry file `chemistry.esh` for non coarse-grained examples contains the following lines

```
ethanol,CCO,20
water,O,80
```

Fraction indications do not have to reflect percentages: 1 instead of 20, combined with 4 instead of 80 is equally valid, due to internal normalization. For a system with 1000 atoms typed with the CHARMM force field and project name `solution`, the call to `emc_setup.pl` is given by

```
emc_setup.pl -ntotal=1000 -field=charmm -replace solution
```

where `-ntotal` indicates the number of atoms, `-field` specifies the desired force field, and `-replace` selects overwriting of possibly existing build and input scripts. An alternative to `chemistry.esh` is presented by `solution.esh`, in which case `emc_setup.pl` command line options are integrated into the script by means of an options section. Execution creates an EMC `build.emc` and a LAMMPS[3,4] `solution.in` script. A change of force field and its corresponding typing occurs by altering the `-field` flag to e.g. `-field=trappe`, when desiring TraPPE typing. Note, that each force field only includes chemistries, for which it was parameterized. This means, that not all chemistries can be typed by all force fields. Also be aware, that typing is not guaranteed to be correct. It is kindly requested to report bugs, so that improvements to force fields can be included in next versions. Next steps for executing EMC and LAMMPS are identical to the steps as mentioned in the previous section for the bulk mixture example.

4.7.2.3 Record

EMC offers the possibility to follow the building process by creating a set of PDB's and accompanying PSF's during building, which is illustrated by the example found in `${EMC_ROOT}/examples/setup/chemistry/record/`. The build of a 60 carbon fullerene has been chosen to illustrate this option. EMC will produce a set of PDB's and PSF's, which are combined by `cat.sh`. The latter produces one PDB and PSF file, and one VMD script, after which it deletes the PDB's and PSF's produced by EMC. A `exec.sh` script illustrates the work flow. It uses `fullerene.esh` as the describing chemistry file, which set the following options,

```
ITEM                OPTIONS

seed                -1
ntotal              60
density             0.1
temperature         300
grace               0.999
nrelax              200
radius              9.5
center              true
field               opls-ua
record              "record",10,true
replace             true
depth               6

ITEM                END
```

A `seed` of -1 indicates for EMC to take the system clock as a seed. The total number of sites is set by `ntotal`, while the `density` is chosen to be 0.1 g/cc. The simulation `temperature` is set at 300 K. The keyword `grace` is used to allow for overlap during initial insertion, which is corrected during relaxation. The number of relaxation moves after insertion, `nrelax`, is set to 200, with a site inclusion `radius` of 0.95 nm. The first inserted site is set at the center of the box by setting keyword `center` to `true`. The chosen force field – as set by defining `field` – is OPLS-UA. Recording is switched on by defining `record`, where PDB's and PSF's are written out for the inserting of each site. The written out files have root file name `"record"`, where each file contains a trajectory resulting from relaxation. Here, snapshots are written every 10 cycles. Inactive sites with positions at (0,0,0) are included in PDB's. All files are opted to be replaced by setting the `replace` keyword to `true`. Ring recognition normally is automatic. However, this would take unnecesarily long for fullerenes due to the recursive nature of the underlying algorithm. EMC allows for reducing the recursion depth of this algorithm. Here, we chose to set the recursion `depth` to 6, since none of the rings in a fullerene have more than 6 members.

Fullerene chemistry is defined in the shorthand paragraph by

```
ITEM                SHORTHAND
```

```
fullerene           c12c3c4c5c1c6c7c8c2c9c1c3c2c3c4c4c%10c5c5c6c6c7c7c%11
                    c8c9c8c9c1c2c1c2c3c4c3c4c%10c5c5c6c6c7c7c%11c8c8c9c1
                    c1c2c3c2c4c5c6c3c7c8c1c23,1

ITEM                END
```

Intricate connectivity is illustrated by connectivity numbering (see Section 7.8 [SMILES], page 197). The SMILES for this example as well as for many chemistries can be found in Wikipedia.[1]

Execution of this example is best performed by workflow script

```
./exec.sh
```

which calls `emc_setup.pl`, followed by EMC. It also concatenates building trajectories as produced by EMC into one set of files, which can be viewed with VMD by

```
vmd -e record.vmd
```

assuming VMD is in the user's path.[2,3]

References

1. N.N., "Buckminsterfullerene", Wikipedia, 1/1/2016.
2. VMD - Visual Molecular Dynamics, '`http://www.ks.uiuc.edu/Research/vmd/`'
3. Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", *J. Molec. Graphics* **1996**, *14*, 33-38

4.7.2.4 Polymers

Polymers have a wide field of application in chemical industry. EMC offers the possibility of creating a variety polymer classes. Examples can be found in `${EMC_ROOT}/examples/setup/chemistry/polymer`. The focus in this section lays on building random, block, and alternate copolymers.

Let us assume, that we would like to study a copolymer consisting of four A beads, four B beads, and two D terminators. We would like to study a coarse-grained model with 1000 beads, using a general DPD force field with a number density of 3.

The `OPTIONS` section is defined by (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩),

```
ITEM        OPTIONS

project     dpd
field       dpd/general
auto        true
ntotal      1000
density     3
replace     true
build_dir   .

ITEM        END
```

The first line sets the project name with keyword `project`, while the second selects the general DPD force field with keyword `field`. The option `auto` toggles the automatic addition of masses to the force field mass paragraph, in case masses are not provided. The total number of beads is set with `ntotal` and the system density with `density`. The keyword `replace` allows for existing files to be overwritten. The current directory is chosen as the build directory by keyword `build_dir`.

The generalized DPD form does not require a `parameters.csv` or `reference.csv` file, when not specifically defining pair-wise interactions. With this information, `emc_setup.pl` will create `dpd.prm` in the current directory, which is a DPD force field parameter file. This force field file is needed by EMC, in order to be able to type the different beads.

A polymer chemistry file cannot use shorthand notation as discussed in the previous paragraphs. Instead, group, cluster, and polymer specific sections need to be used (see Section 4.6.2 [Chemistry File], page 38).

First, three contributing groups are defined to describe beads A, B, and D,

```
ITEM        GROUPS

a           *A*,1,a:2,2,b:1
b           *B*,1,b:2
d           *D,1,a:1,1,a:2,1,b:1,1,b:2

ITEM        END
```

As with the bulk case, SMILES syntax is used to describe group chemistry. Additionally, connectivity between the different monomers needs to be defined. This connectivity is indicates by an asterix `*`. By definition, a monomer has two or more connections and a terminator only one. With the connection points defined per monomer, we have to still define where these connection points connect to. Let us consider group `a` as an example. We would like to connect group `a` to both itself and group `b`,

```
*A*,1,a:2,2,b:1
```

`*A*` indicates the SMILES representation, followed by connection pairs. The first pair `1,a:2` means, that the first occurring `*` of `*A*` connects to the second occuring `*` of group `a`. The first pair `2,b:1` means, that the second occurring `*` of `*A*` connects to the first occuring `*` of group `b`. The same logic applies for subsequent definitions of groups `b` and `d`. The setup script adds all redundant connections, i.e. when connecting group `a` to group `b` with `2,b:1`, a connection on group `b` reading `1,a:2` is implied and does not have to be defined explicitly. One could define group `b` as

```
*B*,1,b:2,2,a:1
```

which would work, but it is not necessary. The defintion of group `d` through

```
*D,1,a:1,1,a:2,1,b:1,1,b:2
```

defines all capping groups to the correct connectors. With the groups defined, we can now define the type of polymer we would like to build,

```
ITEM        CLUSTERS

poly        random,1

ITEM        END
```

The first keyword defines the polymer's name, `poly` here. Its name is followed by the type of polymer. The three available options are `random`, `block`, and `alternate`. Examples are available for each of these options in `${EMC_ROOT}/examples/polymer`. Here we chose to create a random copolymer. The last entry denotes the polymer's fraction, here chosen to be `1`. Different clusters and polymers can be added to this paragraph to form any mixture desired.

Finally, we define the polymer's morphology. We wanted 4 monomers A, 4 monomers B, and two terminators D,

```
ITEM        POLYMERS

poly
1           a,4,b,4,d,2
```

```
ITEM        END
```

Each polymer definition starts with its name as a header on a separate line. This name refers to the name chosen in the clusters section, here `poly`. The polymer's name is followed by all morphologies associated with this name. Here we only have one morphology, but we could also define a distribution of polymer morphologies. Each line contains the mol fraction of each contribution, followed by its morphological definition,

```
1           a,4,b,4,d,2
```

A subsequent polymer would start with its name on a separate line, followed by its definition(s), e.g. this section with two polymers would look like

```
ITEM        POLYMERS

poly
1           a,4,b,4,d,2

another
1           a,2,b,2,d,2
2           a,3,b,3,d,2
1           a,4,b,4,d,2

ITEM        END
```

Here, a polymer called `another` was added and assumed to have three contributing subpolymers, thus creating a distrubtion of polymer lengths within this polymer. The distribution uses mol fractions, which means, that the above example has a ditribution of 1:2:1, or 25% of the first entry, 50% of the second, and 25% of the third. The `emc_setup.pl` script normalizes fractions in the resulting EMC build script.

4.7.2.5 Multiphase Systems

EMC offers the possibility of building multiphase systems by building one phase after another. This allows for setting up systems, for which it is known a priori, that the equilibrated state phase separates. This then allows for studying mass distribution and interfacial tension behavior. Let us investigate this possibility by means of an example, in which we study the behavior of sodium dodecyl sulfate (SDS) in a system of water and oil. This example can be found in `${EMC_ROOT}/examples/setup/chemistry/multiphase`, where the describing file `multiphase.esh` resides.

We assume the density of the water, SDS, and oil phase to be 1, 0.8, and 0.8 g/cc respectively. This system is at a temperature of 300K. We would like to describe the interactions with the CHARMM force field. The final simulation should have about 5000 particles. These prerequisites are captures in the `OPTIONS` paragraph (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩),

```
ITEM        OPTIONS

ntotal      5000
temperature 300
density     1,0.8,0.8
phases      water + sds + oil
field       charmm/c32b1/all_27_prot_lipid
mass        true
profile     true
replace     true

ITEM        END
```

Here, the keyword `ntotal` allows for setting the final number of sites to be simulated. The keyword `temperature` sets the temperature. The desnity of each phase is defined by keyword `density` through separation of each entry by either a comma or a tab. Chemically, each phase is defined by the keyword `phases`. Here, each phase is demarcated by the `+`-sign. Contributing clusters to each individual phase are separated by either commas, tabs, or spaces, e.g.

```
phases      water sds + oil
```

would create two phases: the middle phase would consist of a random mixture of water and sds, while the outer phase would consist of just oil. The fractions entered in the `SHORTHAND` are assumed to be by mass, which is expressed by setting the keyword `mass` to `true` in the `OPTIONS` section. Furthermore, analysis mass profiles is added to the generated LAMMPS input script `multiphase.in` by setting the keyword `profile` to `true`. Lastly, replacement of all scripts and files produced by executing `emc_setup.pl` are selected to be overwritten by setting keyword `replace` to `true`. Defining the chemical composition can be entered in the `SHORTHAND` section (see Section 4.6.2 [Chemistry File], page 38),

```
ITEM        SHORTHAND
```

```
water       O,0.4
oil         (C)8,0.4
sds         CCCCCCCCCCCCO:S(:O)(:O):O.[Na+],0.2

ITEM        END
```

Here, we select 40% by weight of both water and oil phases, while we add 20% by weight of SDS. Chemical identities are entered using the SMILES format (see Section 7.8 [SMILES], page 197).

Steps of execution are described in the usage section (see Section 4.2 [Setup Usage], page 9).

4.7.2.6 Surfaces

EMC allows you to set up surfaces and pack material between them. To this extend, the surface has to be identified by a `surface` keyword in the `CLUSTERS` section of your `emc_setup.pl` script. An example can be found in `${EMC_ROOT}/examples/setup/chemistry/surface`, where `surface.esh` provides the input information and `setup.sh` the manner in which `emc_setup.pl` should be called. In this example, we want to sandwich amorphous 100 decane molecules between two diamond surfaces, where the density of the decane phase should be 0.8 g/cc at a temperature of 298K. We have to first define the simulation conditions in the `OPTIONS` section (see ⟨undefined⟩ [Chemistry Options], page ⟨undefined⟩),

```
ITEM        OPTIONS

density     0.8
temperature 298
ntotal      1000
shape       1.5
build_dir   .
field       opls-ua
charge      false
profile     true
replace     true

ITEM        END
```

Here, we chose the OPLS-UA force field without charges, select a mass profile section to appear in the LAMMPS input script, and allow for `emc_setup.pl` output scripts to be overwritten. The definition of surfaces occurs through predefined EMC input files, of which a few examples can be found in `./lib`. One of these input files is a fully bonded diamond lattice called `diamond_lattice.emc`. EMC input files can contain variables, which can be redefined from an EMC input script. Specific variables need to be redefined, in order to turn the generic diamond lattice input file into a carbon diamond lattice. This redefintion occurs in the `VARIABLES` paragraph (see Section 4.6.2 [Chemistry File], page 38),

```
ITEM        VARIABLES

diamond_chemistry   "*C(*)(*)*"
diamond_name        "carbon"
diamond_atomistic   true

lbond               1.529

ITEM        END
```

Four variables are predefined in the diamond lattice input file: `diamond_chemistry`, `diamond_name`, `diamond_atomistic`, and `lbond`. The above defintions overwrite the

predefined values. The value for `lbond` has been taken from the OPLS-UA force field, where it represents the equilibrium bond distance between two `ct` types, which is 1.529 A. With options and variables set, we now need to define the group describing decane,

```
ITEM        GROUPS

decane      (C)10

ITEM        END
```

Here, an extension of SMILES was used to depict decane. Proper SMILES would require `CCCCCCCCCC`, but EMC allows for an extension to SMILES, for which a number after parenthesis is interpreted as the number of times, with which the snippet between the parenthesis is repeated. Finally, the `CLUSTERS` cluster section defines both decane and the surface,

```
ITEM        CLUSTERS

decane      decane          1
surface     import          4       $root+"lib/diamond_bonded"      emc

ITEM        END
```

Here, the fraction, 1, of decane molecules is defined, followed by the surface definition. Both lines start with the name of the cluster. This name is followed by a reference to the corresponding group for the decane molecules. The second keyword for the surface molecule is a reserved keyword, which indicates, that this molecule is a surface, which uses the library file `diamond_bonded` as a reference file. The different phases are built in the x-direction. The number `4` indicates the number of surface cells to build into this direction, which defines the value of internal variable `nx`.

4.7.3 Environment Mode

The environment mode allows creating a multi-simulation environment, conveniently packaged in one clear file. One example dealing with determining shear is included. Environment examples can be found in `${EMC_ROOT}/examples/setup/environment/`.

4.7.3.1 User-Defined Force Fields

This example demonstrates the concept of including a force field definition in a chemistry template, while adding loops in the environment section of the EMC Setup input script. The example can be found in `${EMC_ROOT}/examples/setup/environment/field`. EMC Setup provides the `FIELD` section for controlling user-defined force fields. The idea behind this example is to study the hypothetical influence of branching. To this extend, the number of branch points and the Lennard-Jones (LJ) interaction constant epsilon is varied. The work directory `${WORKDIR}` is defined by

```
${EMC_ROOT}/examples/setup/environment/field
```

later referred to by `${WORKDIR}`. The user is advised to maintain a standardized directory structure when using the environment mode of EMC Setup. This means, that EMC Setup environment .esh files are stored in `${WORKDIR}`/setup. Execution of the setup files should occur in `${WORKDIR}` and not in `${WORKDIR}/setup`. Once executed in `${WORKDIR}`, EMC Setup creates directories `analyze`, `build`, `chemistry`, `run`, and optionally `test` in `${WORKDIR}`. A typical convention for setup files is to name them by date, followed by a serial number, e.g. `2018070100` means, that the setup file was created on July 1, 2018 and is the first of the series of files created on that date.

Environment

All sections outside the `TEMPLATE` section define the environment and is considered to be the environment mode of EMC Setup. This includes the `ENVIRONMENT` and `LOOPS` sections as well as the `STAGE` designator. The `ENVIRONMENT` section defines the project name, queue settings, and high performance computing (HPC) cluster architecture parameters. Additionally, the name of a test directory can be defined by the option `name_testdir` (see the 'Execution' section below).

Loops

Loop variables are defined in the `LOOPS` section. Variables `stage` and `trial` are reserved keywords. The former indicates the stage of a project, e.g. one likes to study branched and linear structures. This could be devided in a `branched` stage and a `linear` stage. Here, only the `branched` variety is considered. Furthermore, the number of branches is indicated by `nbranches` and the settings for the LJ interaction constant by `epsilon`. These variables are later referred to in the `TEMPLATE` section by `@{NBRANCHES}` and `@{EPSILON}` respectively.

Template

The `TEMPLATE` section contains the definition of one simulation, here referred to as the chemistry mode of EMC Setup. In this chemistry template, one can use variables as defined in the `LOOPS` section of the environment. A template should at least contain an `OPTIONS`, `GROUPS`, and `CLUSTERS` section to fully define the simulation setup. Here, an additional `POLYMERS` section has been added to control the definition of the to be studied branched oligomers. Since polymers are requested, connectivity has to be defined in the `GROUPS` section. This connectivity is referred to with the '*' character. The numbers after the

chemical definition through a SMILES refer to the occurence of the '*' character in the SMILES, e.g. the definition of group `A`

```
*a*,1,A:2,1,B:2,1,B:3,2,B:1
```

means, that the first '*' in '*a*' connects to the second of group `A` and the second and third of group `B`. The second star in '*a*' connects the first of group `B`.

The `CLUSTERS` section defines the molecule and the amount of each contributor, i.e. the solvent is defined by group S and has an 80% occurence. The polymer is defined as a random copolymer and has a 20% occurence.

The `POLYMERS` paragraph holds all definitions of polymers as referred to in the `CLUSTERS` paragraph. Here, only the polymer `polymer` exists with 10 repeat units of monomer A as defined by group A, and a variable amount of monomer B and terminator T. The amount is defined by `@{NBRANCHES}` which refers to `nbranches` in the `LOOPS` section.

The `FIELD` section defines the user-provided force fields and contains subsections related to the subsequent parameter definitions. Mandatory subsections are `MASS`, `NONBOND`, and – in case of connectivity – `BOND`. Additionally, in the case of this example angle terms are also defined in the `ANGLE` subsection. No torsions were needed. Wildcards in the form of '*' are allowed in all but the MASS section. In the case of the latter, each type can only contain one '*' character, which has to be at the end of the type. Note that the variable definition of the LJ epsilon is taken care of by `@{EPSILON}` in the `NONBOND` subsection.

Execution

Creation of the simulation environment occurs in `${WORKDIR}` through

```
./setup/2018070100.esh
```

or, when the above is not executable or `emc_setup.pl` is not in your path,

```
${EMC_ROOT}/scripts/emc_setup.pl setup/2018070100.esh
```

Though the latter being possible, the user is adviced to add both `bin` and `scripts` directory to their path, e.g.

```
export PATH=${EMC_ROOT}/bin:${EMC_ROOT}/scripts:${PATH}
```

Once executed, several bash scripts are generated in directories `analyze`, `build`, and `run` with name `2018070100.sh`. Subsequently, the provided test directory name appears as a subdirectory of `${WORKDIR}/test/`, in which the script `setup.sh` is created. Execution of the latter creates an instance of the first occurence of each loop variable, i.e. for `STAGE = branched`, `NBRANCHES = 2`, and `EPSILON = 0.9`. This test directory serves the purpose of testing the validity of the `TEMPLATE` section without having to submit to a queueing system.

Before summitting, an option which you would need to set is queue_ppn (procs or cores per node), since it is unknow how many cores can run on one of your nodes. EMC will

pack single core build jobs together on a node if it knows the ppn. Similarly, you can set the memory per core with queue_memory.

Submission to a queueing system occurs though `./build/2018070100.sh`, followed by `./run/2018070100.sh` once all builds as spawned by the former have finished. Chaining of builds and runs is also possible by executing the run script with build mode included, i.e.

```
./run/2018070100.sh -build
```

This spawns build and lets running of the systems LAMMPS wait for the building to finish.

4.7.3.2 Shear

The shear example, as found in `${EMC_ROOT}/examples/setup/environment/nemd`, illustrates the use of the EMC setup environment mode to define a set of simulations, calculating time dependent pressure tensors for a number of n-alkanes with various shear rates. The enviroment mode allows for scanning through parameter space, aiding the study of material response to various conditions. The example defines 40 simulations in total. An environment script contains multiple distinct sections. A shear simulation needs an equilibrated non-sheared configuration as a starting point. First, the environment mode needs to be triggered in the first occurring `OPTIONS` section,

```
ITEM        OPTIONS

project             pure
replace             true
environment         true
name_analyze        date00
name_build          date00
name_run            date00
queue_build         default
queue_analyze       default
shear               true, true
restart             fal
ncores              8
trun                -

ITEM        END
```

Besides toggling the environment mode, this section also defines the general project name, names associated with build, run, and analysis scripts, execution queues, and the number of cores to execute LAMMPS simulations on. Build, run, and analysis names use the base name of the chemistry script by as default. Zero shear simulations do not start from a restart file, but uses the output generated by EMC instead.

Loops can be defined over project stages, trials, and user defined variables in section `LOOPS`. This section controls parameter space for which phenomena are to be studied.

```
ITEM        LOOPS

stage               pure
trial               c-06    c-07    c-08    c-09    c-10    &
                    c-11    c-12    c-13    c-14    c-16
shear               0.00e0
copy                1

ITEM        END
```

Reserved variables `stage`, `trial`, and `copy` are internally defined. Both `stage` and `trial` relate to other sections in the the environment script. Each `stage` is associated with a specific chemistry file, which is defined in section `TEMPLATE` of the environment script. The definitions for variable `trial` are associated with section `STRUCTURES`. The reserved variable `copy` supports statistical sampling by creating independent structures (here, as an example, 1 structure is created). The use of the `copy` variable should be used with care, since it functions as a multiplier for the number of simulations, that are considered. In this example, the defined 11 trials and 4 shears comprise 44 simulations. This would increase to 440 simulations, when `copy` is set to 10 instead of 1.

The variable `shear` is a user defined variable. It is expected to be lower case. Its reference in the chemistry template is expected to be capitalized. Shell scripts are created after each `LOOPS` section. Build and run scripts – with names `date00.sh` and `date10.sh`, respectively, as defined in the previous `OPTIONS` section – are written in a newly created directory `run` in the current work directory. An analysis script with the name `date10.sh` is created in in a newly created directory `analyze`.

Subsequent shear simulations are defined after zero shear simulations. Here, only a few lines are needed. Non-defined parameters take the value of their definition in previous `OPTIONS` sections.

```
ITEM        OPTIONS

name_build          -
name_run            date10
name_analyze        date10
restart             true, ../../0.00e0

ITEM        END
```

These subsequent simulations will use equilibrated zero shear simulations as their starting point. A build script is not needed in this case, which is denoted by defining parameter `name_build` with a dash. Restart files are used as starting point, which is triggered by setting parameter `restart` and defining the source directory for the restart files, `../../0.00e0` in this case (the `0.00e0` value refers to the previously set zero shear).

Multiple shear rates are needed to see the shear behavior, intending to reach the Newtonian plateau at very low shear. Depending on the substance, this plateau might not be possible to be reached.

```
ITEM        LOOPS

shear               1.00e-4, 1.00e-5, 1.00e-6

ITEM        END
```

Shear rates are expressed in box length per time step. After the definition of this paragraph, a run shell script with name `2016103120.sh` is added to directory `run`, and an analysis shell

script with name `2016103120.sh` is added to directory `analyze`. Note, that the definition of `stage` and `trial` are inherited from the previous definition of section `LOOPS`.

With all shell scripts defined, now we can define each stage and trial separately by using sections `STAGE` and `TRIAL` respectively. After invoking `STAGE` and/or `trial`, all subsequent sections apply to the defined stage and/or trial. Here, we only need to define a section `STAGE` with name `pure`, relating back to the definition of variable `stage` in section `LOOPS`. Note, that both sections `STAGE` and `TRIAL` do not require an `ITEM END` designation.

```
ITEM        STAGE   pure
```

A template is used to define the actual simulation conditions. A template section is defined by `ITEM TEMPLATE` and needs an `ITEM END` to close. Variables appearing in section `LOOPS` can be referred to by starting with `@` followed by the variable name in all capitals. A simulation is defined as usual withing a template section.

OPLS-UA reflects the force field chosen for our n-alkanes. A total of 10000 beads is requested by setting `ntotal`. The equilibration time is set to 100000 time steps with `tequil`. The initial `density` is chosen to be 0.85 g/cc. The simulation is run at equal `pressure` of 1 atm and `temperature` of 300 K by using an NPT ensemble. Note, that it is possible, that `npt/sllod` is not available in all LAMMPS versions, in which case the presure line should be omitted. Shear rates are set by the `shear` parameter. Here, `@SHEAR` refers back to variable `shear` in the `LOOPS` section. All exisiting scripts are overwritten by setting parameter `replace` to `true`. The usage of bond increments depend on the chosen force field. An empty bond increment is used when non-existing bond increments are encountered. Note, that OPLS-UA does not use bond increments.

```
ITEM        OPTIONS

replace             true
field               opls-ua
ntotal              10000
tequil              100000
density             0.85
pressure            1.0
temperature         300
shear               @SHEAR,erate
replace             true
increment           empty

ITEM        END
```

With simulation conditions set, substance definitions are in order. We chose to use a structure section to define chemical structures. As a consequence, the variable `structure` is defined and used to replace all occuring `@STRUCTURE` instances in the chemistry template.

```
ITEM        SHORTHAND
```

```
molecule            @STRUCTURE,1

ITEM        END
```

The template section is closed with an `ITEM END`. This template is used to define all simulations. It is stored with name `pure.esh` in a created directory `chemistry/stages` within the current work directory. Also, an adaptation shell script with name `pure.sh` is stored in created directory `chemistry/scripts`.

Finally, desired substances have to be defined. Doing so in section `STRUCTURES` suffices, since we are only interested in bulk properties of pure components.

```
ITEM        STRUCTURES

c-06                (C)6
c-07                (C)7
c-08                (C)8
c-09                (C)9
c-10                (C)10
c-11                (C)11
c-12                (C)12
c-13                (C)13
c-14                (C)14
c-15                (C)15
c-16                (C)16

ITEM        END
```

Each structure is stored in a separate file with names as occuring in the first column. These files are stored in directory `chemistry/structures`.

Execution of the whole simulation can be done on either the cluster on which the simulations are to run, or can be done from the machine on which the final results are to be analyzed. In the former case, each of the three scripts in directories `build`, `run`, and `analysis` are to be executed sequentially, i.e. `build/date00.sh` to build all simulations, subsequently followed by zero shear simulations through `run/date00.sh` when all build simulations are completed, an by finite shear simulations through `run/date10.sh` when all zero shear simulations are completed. Each script uses `run.sh` to submit jobs to the queueing system, which can be found in `${EMC_ROOT}/scripts`. The `run.sh` supports LSF and PBS queueing systems. The `run_host.sh` script can be found in the same directory allows remote submission of jobs. It assumes, that the execution directory structure is identical on both local and remote host machines. `run_host.sh` can be used to for instance execute the build shell script remotely through execution of

```
run_host.sh user@remote.machine.net build/date00.sh
```

in the current work directory. Log files are created both locally and remotely by `run_host.sh`, keeping records of simulation execution. The example above would create

`build/date00.log`. Remote execution of subsequent run scripts would follow each other sequentially upon completion of its predecessor.

4.8 Help Output

The following output appears when invoking `emc_setup.pl` either without any options, with a non-exisiting option, or with the `-help` option. Additionally, using keyword `module` adds the originating module to the output (i.e. `-help=module`).

```
EMC Setup v5.0 (January 1, 2023), (c) 2004-2023 Pieter J. in 't Veld

Usage:
  emc.pl [-command[=#[,..]]] project [phase 1 clusters [+ ...]]

Commands:
  -analyze_archive      archive file names associated with analyzed data
                        [true]
  -analyze_data         create tar archive from exchange file list [true]
  -analyze_last         include last trajectory frame (deprecated) [false]
  -analyze_replace      replace already exisiting analysis results [true]
  -analyze_skip         set the number of initial frames to skip [0]
  -analyze_source       set data source directory for analysis scripts
  -analyze_user         set directory for user analysis scripts
  -analyze_window       set the number of frames in window average [1]
  -angle                set DPD angle constants k and theta or set angle
                        field option (see below) [5,180]
  -auto                 add wildcard entry to mass and nonbond sections in
                        DPD .prm [false]
  -binsize              set bin size for LAMMPS profiles [0.01]
  -bond                 set bond constants k,l
  -build                set build script name [build]
  -build_center         insert first site at the box center [false]
  -build_dir            set build directory for LAMMPS script [../build]
  -build_order          set build order of clusters [random]
  -build_origin         set build order of clusters [x=0, y=0, z=0]
  -build_replace        replace already existing build results [false]
  -build_theta          set the minimal insertion angle [false]
  -charge               chemistry contains charges [true]
  -charge_cut           set charge interaction cut off [9.5]
  -chunk                use chunk approach for profiles in LAMMPS script
                        (depricated, use lammps_chunk) [true]
  -communicate          use communicate keyword in LAMMPS script (depricated,
                        use lammps_communicate) [false]
  -core                 set core diameter [-1]
  -cross                include nonbond cross terms in LAMMPS params file
                        [false]
  -crystal              treat imported structure as a crystal [auto]
  -cut                  set pairwise interaction cut off [9.5]
  -cutoff               generate output of pairwise cut off in LAMMPS
                        parameter file (depricated, use lammps_cutoff)
                        [false]
```

```
-debug              output debugging information [false]
-deform             deform system from given density [nblocks=1,
                    ncycles=100, type=relative, xx=1, yx=0, yy=1, zx=0,
                    zy=0, zz=1]
-delete             sets which clusters to delete; each deletion is
                    separated by a +-sign; default assigns no clusters to
                    delete [clusters=all, fraction=1, groups=all,
                    mode=include, phase=1, sites=all, thickness=1,
                    type=relative]
-density            set simulation density in g/cc for each phase [1]
-dielectric         set charge medium dielectric constant [1]
-direction          set build direction of phases [x]
-dlimit             set LAMMPS nve/limit distance (depricated, use
                    lammps_dlimit) [0.1]
-dtdump             set LAMMPS trajectory file write frequency
                    (depricated, use lammps_dtdump) [100000]
-dtrestart          set LAMMPS restart file frequency (depricated, use
                    lammps_dtrestart) [100000]
-dtthermo           set LAMMPS thermodynamic output frequency
                    (depricated, use lammps_dtthermo) [1000]
-emc                create EMC build script [true]
-emc_depth          set ring recognition depth in groups paragraph [8]
-emc_exclude        set EMC section to exclude [build=false]
-emc_execute        execute EMC build script [false]
-emc_export         set EMC section to export [smiles=]
-emc_moves          set Monte Carlo moves for after build
                    [cluster=HASH(0x7fd741241c28),
                    displace=HASH(0x7fd741241ce8)]
-emc_output         set EMC output modes [debug=false, exit=true,
                    info=true, warning=true]
-emc_progress       set progress indicators [build=true, clusters=false]
-emc_run            set Monte Carlo run conditions for after build
                    [clusters=all, groups=all, nblocks=100, ncycles=0,
                    nequil=0, sites=all]
-emc_test           test EMC build script [false]
-emc_traject        settings for EMC trajectory [append=true,
                    frequency=0]
-environment        create project environment [false]
-error              restart LAMMPS only upon previous error (depricated,
                    use lammps_error) [false]
-ewald              set long range ewald summations [true]
-exclude            exclude previous phase during build process [true]
-expert             set expert mode to diminish error checking [false]
-extension          set environment script extension [.esh]
-field              set force field type and name based on root location
-field_angle        set angle field option (see below) [-]
-field_bond         set bond field option (see below) [-]
```

```
-field_charge         check system charge after applying force field [true]
-field_check          check force field compatibility [true]
-field_debug          set debug field option [false]
-field_dpd            set various DPD options [auto=false, bond=false]
-field_error          override field errors (used for debugging) [true]
-field_format         parameter format of generated force field [%15.10e]
-field_group          set group field option (see below) [-]
-field_id             set force field id [opls-ua]
-field_improper       set improper field option (see below) [-]
-field_increment      set increment field option (see below) [-]
-field_location       set force field location [~/emc/v9.4.4/field]
-field_name           set force field name
-field_nbonded        set number of excluded bonded interactions [0]
-field_reduced        set force field reduced units flag [false]
-field_torsion        set torsion field option (see below) [-]
-field_type           set force field type [opls]
-field_write          create field parameter file [true]
-focus                list of molecules to focus on [-]
-ghost_cut            set pairwise interaction cut off [-1]
-grace                (deprecated: use weight) set build relaxation grace
                      [0.9999,0.9999,0]
-help                this message
-host                set host on which to run simulations [pro]
-info                output standard information [true]
-inner               set inner cut off [-1]
-insight              create InsightII CAR and MDF output [false]
-insight_compress     set InsightII CAR and MDF compression [false]
-insight_pbc          apply periodic boundary conditions [false]
-insight_unwrap       apply unwrapping [false]
-kappa                set electrostatics kappa [4]
-lammps               create LAMMPS input script or set LAMMPS version
                      using year, e.g. -lammps=2014 (new versions start at
                      2015) [true]
-lammps_chunk         use chunk approach for profiles in script [true]
-lammps_communicate   use communicate keyword in script [false]
-lammps_cutoff        generate output of pairwise cut off in parameter file
                      [false]
-lammps_dlimit        set nve/limit distance [0.1]
-lammps_dtdump        set trajectory file write frequency [100000]
-lammps_dtrestart     set restart file frequency [100000]
-lammps_dtthermo      set thermodynamic output frequency [1000]
-lammps_error         restart only upon previous error [false]
-lammps_momentum      set zero total momentum [100,1,1,1,angular]
-lammps_nsample       number of configuration in profile [1000]
-lammps_pdamp         set barostat damping constant [1000]
-lammps_prefix        set project name as prefix to output files [false]
-lammps_skin          set skin [2]
```

```
-lammps_tdamp          set thermostat damping constant [100]
-lammps_tequil         set equilibration time [1000]
-lammps_tfreq          set profile sampling frequency [10]
-lammps_thermo_multi   set thermo style to multi [false]
-lammps_triclinic      set triclinic mode [false]
-lammps_trun           set run time [10000000]
-location              prepend paths for various file locations
                       [analyze=.:~/emc/v9.4.4/scripts/analyze,
                       field=.:~/emc/v9.4.4/field, include=.]
-mass                  assume mass fractions in chemistry file [false]
-md_restart            create MD restart script [false,..]
-md_shake              set shake types
-md_shake_iterations   set maximum number of shake iterations [20]
-md_shake_output       set shake output frequency [never]
-md_shake_tolerance    set shake tolerance [0.0001]
-md_shear              add shear paragraph to LAMMPS input script [false]
-md_timestep           set integration time step [2]
-memorypercore         set queue memory per core in gb [default]
-modules               manipulate runtime modules in format [command=]module
-mol                   assume mol fractions in chemistry file [true]
-momentum              set zero total momentum in LAMMPS (depricated, use
                       lammps_momentum) [100,1,1,1,angular]
-moves_cluster         define cluster move settings used to optimize build
                       [active=false, cut=0.05, frequency=1,
                       limit=auto:auto, max=0:0, min=auto:auto]
-msd                   set LAMMPS mean square displacement output [false]
-namd                  create NAMD input script and parameter file [false]
-namd_dtcoulomb        set electrostatic interaction update frequency [1]
-namd_dtdcd            set output frequency of DCD file [10000]
-namd_dtnonbond        set nonbonded interaction update frequency [1]
-namd_dtrestart        set output frequency of restart files [100000]
-namd_dtthermo         set output frequency of thermodynamic quantities
                       [1000]
-namd_dttiming         set timing frequency [10000]
-namd_dtupdate         set update frequency [20]
-namd_extra_bonds      set selection for extra bonds [-]
-namd_fixed_atoms      set selection for fixed atoms [-]
-namd_pres_decay       set pressure ensemble decay [50]
-namd_pres_period      set pressure ensemble period [100]
-namd_temp_damp        set temperature ensemble damping [3]
-namd_tequil           set number of equilibration timesteps [100000]
-namd_tminimize        set number of minimization timesteps [50000]
-namd_trun             set number of timesteps for running [10000000]
-name_analyze          set job analyze script name [chemistry]
-name_build            set job build script name [chemistry]
-name_run              set job run script name [chemistry]
-name_scripts          set analyze, job, and build script names
```

```
                    simultaneously
-name_testdir       set job test directory as created in ./test/ [-]
-nchains            set number of chains for execution of MD jobs
-ncores             set number of cores for execution of MD jobs [-1]
-ncorespernode      set queue cores per node for packing jobs [default]
-niterations        set number of build insertion iterations [1000]
-norestart          control possibility of restarting when rerunning
                    [false]
-nparallel          set number of surface parallel repeat unit cells
                    [auto]
-nrelax             set number of build relaxation cycles [100]
-nsample            number of configuration in profile (depricated, use
                    lammps_nsample) [1000]
-nthreads           set number of cores for per thread for MD jobs [1]
-ntotal             set total number of atoms [10000]
-number             assume number of molecules in chemistry file [false]
-omit               omit fractions from chemistry file [false]
-options_perl       export options, comments, and default values in Perl
                    syntax [false]
-options_tcl        export options, comments, and default values in Tcl
                    syntax [false]
-outer              set outer cut off [-1]
-pair               set pair constant defaults [a=25, cutoff=1,
                    gamma=4.5, r=1]
-parameters         set parameters file name [parameters]
-params             create field parameter file [true]
-pdamp              set LAMMPS barostat damping constant (depricated, use
                    lammps_pdamp) [1000]
-pdb                create PDB and PSF output [true]
-pdb_atom           set atom name behavior [index]
-pdb_compress       set PDB and PSF compression [true]
-pdb_connect        add connectivity information [false]
-pdb_cut            cut bonds spanning simulation box [false]
-pdb_extend         use extended format for PSF [false]
-pdb_fixed          do not unwrap fixed sites [true]
-pdb_hexadecimal    set hexadecimal index output in PDB [false]
-pdb_parameters     generate NAMD parameter file [false]
-pdb_pbc            apply periodic boundary conditions [true]
-pdb_rank           apply rank evaluation for coarse-grained output
                    [false]
-pdb_residue        set residue name behavior [index]
-pdb_rigid          do not unwrap rigid sites [true]
-pdb_segment        set segment name behavior [index]
-pdb_unwrap         apply unwrapping [clusters]
-pdb_vdw            add Van der Waals representation [false]
-percolate          import percolating InsightII structure [false]
-phases             sets which clusters to assign to each phase; each
```

```
                    phase is separated by a +-sign; default assigns all
                    clusters to phase 1 [all]
-polymer            default polymer settings for groups [bias=none,
                    connect=ARRAY(0x7fd74123c068), fraction=number,
                    niterations=-1, order=list]
-polymer_niters     number of iterations for polymer construction [-1]
-port               port EMC setup variables to other applications
-precision          set charge kspace precision [0.001]
-prefix             set project name as prefix to LAMMPS output files
                    (depricated, use lammps_prefix) [false]
-preprocess         use gcc to preprocess the input script [false]
-pressure           set system pressure and invoke NPT ensemble;
                    optionally add direction and/or (un)couple for
                    specifying directional coupling [false,
                    direction=x+y+z, couple]
-profile            set LAMMPS profile output [density=false,
                    density3d=false, pressure=false]
-project            set project name; slashes are used to create
                    subdirectories
-queue              queue settings [account=none, analyze=default,
                    build=default, memory=default, ncores=-1,
                    ppn=default, ppt=1, run=default, user=none]
-queue_account      set queue account for billing [none]
-queue_analyze      set job analyze script queue [default]
-queue_build        set job build script queue [default]
-queue_memory       set queue memory per core in gb [default]
-queue_ncores       set number of cores for execution of MD jobs [-1]
-queue_ppn          set queue cores per thread [1]
-queue_run          set job run script queue [default]
-queue_user         options to be passed directly to queuing system
                    [none]
-quiet              quiet output [false]
-radius             set build relaxation radius [5]
-record             set record entry in build paragraph [cut=false,
                    frequency=1, inactive=true, name="", pbc=true,
                    unwrap=sites]
-references         set references file name [references]
-region_epsilon     set epsilon to use for exclusion regions [0.1]
-region_sigma       set sigma to use for exclusion regions [1]
-replace            replace all written scripts as produced by EMC setup
                    [false]
-restart            create MD restart script (depricated, use md_restart)
                    [false,..]
-rlength            set reference length
-rmass              set reference mass
-rmax               set maximum build cutoff radius [-1]
-rtype              set reference type
```

```
-sample              set sampling options [energy=false, green-kubo=false,
                     gyration=false, msd=false, pressure=true,
                     volume=false]
-script              set script file name [chemistry]
-script_ncolums      set number of colums in output scripts [80]
-seed                set initial random seed [-1]
-shake               set shake types (depricated, use md_shake)
-shake_iterations    set maximum number of shake iterations (depricated,
                     use md_shake_iterations) [20]
-shake_output        set shake output frequency (depricated, use
                     md_shake_output) [never]
-shake_tolerance     set shake tolerance (depricated, use
                     md_shake_tolerance) [0.0001]
-shape               set shape factor [1]
-shear               add shear paragraph to LAMMPS input script
                     (depricated, use md_shear) [false]
-skin                set LAMMPS skin (depricated, use lammps_skin) [2]
-split               sets which clusters to partition; each split is
                     separated by a +-sign; default assigns no clusters to
                     split [clusters=all, fraction=0.5, groups=all,
                     mode=random, phase=1, sites=all, thickness=1,
                     type=relative]
-suffix              set EMC and LAMMPS suffix [_pro]
-system              system identification and checks during building
                     [charge=true, geometry=true, id=main, map=true,
                     pbc=true]
-system_charge       check for charge neutrality after build [true]
-system_geometry     check geometry sizing upon building [true]
-system_id           check for charge neutrality after build [main]
-system_map          map system box before build [true]
-system_pbc          apply periodic boundary conditions after build [true]
-tdamp               set LAMMPS thermostat damping constant (depricated,
                     use lammps_tdamp) [100]
-temperature         set simulation temperature [300]
-tequil              set LAMMPS equilibration time (depricated, use
                     lammps_tequil) [1000]
-tfreq               set LAMMPS profile sampling frequency (depricated,
                     use lammps_tfreq) [10]
-thermo_multi        set LAMMPS thermo style to multi (depricated, use
                     lammps_thermo_multi) [false]
-tighten             set tightening of simulation box for imported
                     structures [3]
-time_analyze        set job analyze script wall time [00:30:00]
-time_build          set job build script wall time [00:10:00]
-time_run            set job run script wall time [24:00:00]
-timestep            set integration time step (depricated, use
                     md_timestep) [2]
```

```
  -trace                provide function trace upon error [false]
  -triclinic            set LAMMPS triclinic mode (depricated, use
                        lammps_triclinic) [false]
  -trun                 set LAMMPS run time (depricated, use lammps_trun)
                        [10000000]
  -types                output types only [false]
  -units                set units type [real]
  -units_energy         set units for energetic scale [-1]
  -units_length         set units for length scale [-1]
  -version              output version information [false]
  -volume               set recalculation based on molecular volume [false]
  -wall                 set temporary exclusion wall thickness [10]
  -warn                 output warnings [true]
  -weight               set build relaxation energetic weights [bond=0.0001,
                        focus=1, nonbond=0.0001]
  -width                use double width in scripts [false]
  -workdir              set work directory [~/emc/v9.4.4/texinfo/setup]

Notes:
  * This script comes with no warrenty of any kind. It is distributed under
    the same terms as EMC, which are described in the LICENSE file included
    in the EMC distribution.
  * A '+' sign demarcates clusters for each phase; remaining clusters are
    assigned to the first empty phase
  * Chemistry and environment file names are assumed to have .esh extensions
  * File names with suffixes _chem can be taken as chemistry file names wild
    cards
  * Reserved environment loop variables are: stage, trial, and copy
  * Densities for multiple phases are separated by commas
  * Shears are defined in terms of erate; values < 0 turns shear off
  * Inner and outer cut offs are interpreted as fractions for colloidal force
    fields
  * Valid field options are: ignore, complete, warn, empty, or error
  * Queue name 'default' refers to whichever queue is default; queue name
    'local' executes all jobs sequentially on local machine
  * Reserved environment loop variables are: stage, trial, and copy
  * Reference and parameter file names are assumed to have .csv extensions
```

5 Scripting Commands

EMC is driven through a scripting language that accesses the different predefined routines driving building and simulation of configurations. EMC requires each script to start with the text `"(* EMC: Script *)"` on the first line. Absence of this line will result in an error upon start up of EMC. The following sections describe the syntax of commands in the current version. Currently, commands use a variable representation similar to Mathematica. A change to a function representation is planned in future.

5.1 Build

5.1.1 Syntax

```
build           = {
  system        -> {id -> constant, temperature -> real,
                    density -> real, geometry -> voigt,
                    flag -> {charge -> boolean, map -> boolean}},
  ...
  select        -> {progress -> boolean, frequency -> long,
                    message -> option, center -> boolean, origin -> vector,
                    order -> constant, check -> boolean,
                    cluster -> {constant, ...}, name -> string,
                    relax -> {ncycles -> long, radius -> real},
                    grow -> {method -> constant, check -> constant,
                      cutoff -> real, grace -> {real, real},
                      theta -> real, dphi -> real, nbonded -> long,
                      ntrials -> long, niterations -> long,
                      include -> {region, ...},
                      exclude -> {region, ...}}}
  ...
};
```

Directive	Parameters	Description
`system`	`struct`	Defines the system dimensions associated with a build.
`id`	`constant`	Sets identity of system to which to add the build; can be either textual or numerical.
`density`	`real`	Sets the system density; can either be set to `calculate` when chosing or forced when chosing values > 0.0.
`geometry`	`voigt`	Sets the system geometry; used as a fractional size when `density` is set, otherwise used as system dimensions from which a density is calculated (see Section 7.15 [Voigt], page 367).
`flag`	`boolean`	Control system checks and optimizations.
`charge`	`boolean`	Perform a total charge check of the system and exit on a non-zero value; options are `true` or `false`.
`map`	`boolean`	Map a stretched triclinic cell geometry into its minimal representation; options are `true` or `false`.

Directive	Parameters	Description
`select`	`struct`	Selects building style and clusters
`progress`	`boolean`	Shows progress indicator; options are `bar`, `list`, or `none`.
`frequency`	`long`	Sets the progress frequency (in percentages), where 0 <= frequency <= 100.

`message`	`option`	Sets the style for energetic output in progress report; valid options are `none`, `raw`, `nkt`, or `n`, which corresponds to energy in no particular units, internal units, units of nkT or units of n, respectively; internal units correspond to the units of the selected force field (kcal/mol for COMPASS, OPLS, GROMACS and MARTINI).
`center`	`boolean`	Place first inserted site at origin as defined `origin` option; options are `true` or `false`.
`origin`	`{0,0,0}`	Define origin for first inserted site; used only when `center` option is set to `true`.
`order`	`constant`	Insert clusters in `sequence` or at `random`.
`check`	`boolean`	Check for (semi)fixed sites and abort if present; options are `true` or `false`.
`cluster`	`array`	Sets the cluster(s) participating in the build.
`name`	`string`	Sets the output name for generation of an XYZ file upon error of all active sites of the built system; no output is generated, when the output name is not set.
`relax`	`struct`	Selects intermediate structure relaxation during the building process; Monte Carlo is performed on a spherical volume with a `radius` around the inserted site for `ncycles` after each successful insertion.

Directive	**Parameters**	**Description**
`grow`	`struct`	Describes growth method and settings.
`method`	`option`	Sets building style; options are `overlap` for hard sphere excluded volume considerations, `energetic` for energetic considerations, and `minimum` for energetic minimum considerations.
`check`	`option`	Sets overlap checking; options are `bonded` for checking overlaps up to four bonds away, `cluster` for checking intramolecular, `all` for intra- and intermolecular overlaps, and `none` for no overlap checking.
`cutoff`	`real`	Sets the energetic cutoff when using `method -> minimum`, where decisions are based on boltzmann sampling when `cutoff` <= 0.
`grace`	`{real, real}`	Sets the grace fraction, which correlates to how much overlap is allowed; 0 corresponds to no overlap, while 1 corresponds to full overlap (typically < 0.75); the first value influences the nonbonded acceptance, while the second value influences the bonded acceptance; an energetical building style requires grace < 0.9995 for coherent energies.
`theta`	`real`	Sets the minimum allowed bond angle.

`dphi`	`real`	Sets the maximum allowed torsion angle for for bonded construction using a Monte Carlo scheme; needed for all force field definitions beyond bond length interactions (i.e. it is neglected in case of the existence of bonds only or when a lower threshold is crossed).
`nbonded`	`integer`	Sets the number of trials moves for bonded construction using a Monte Carlo scheme.
`ntrials`	`integer`	Sets the number of trial overlap or energetic calculations.
`niterations`	`integer`	Sets the number of iterative trials.
`include`	`region`	Sets inclusion regions (see Section 7.7 [Region], page 196).
`exclude`	`region`	Sets exclusion regions (see Section 7.7 [Region], page 196).

5.1.2 Usage

The `build` command enables the build of previously defined clusters (see Section 5.4 [Clusters], page 93) into a desired system. All building modes apply energetic sampling of bonded potentials during the bond building process. Currently a hard sphere `overlap` and an `energetic` build is available. Note that a system needs to be defined before build mode can be executed (e.g. define `system` before `select`). Note that `hard` and `soft` modes regulate insertion behavior with respect to exclusion regions. The `hard` mode applies the region to all sites of inserted clusters, while the `soft` mode only applies the region to the first site of inserted clusters.

5.1.3 Default

The default is given by

```
build           = {
  system        -> {id -> 0, density -> calculate, geometry -> {1, 1, 1},
                    flag -> {charge -> true, map -> true}},
  select        -> {progress -> bar, frequency -> 5, message -> nkt,
                      center -> false, order -> random, check -> true,
                      relax -> {
                        ncycles -> 0},
                      grow -> {
                        method -> energetic, check -> all,
                        grace -> {0.99, 0}, theta -> 0.0,
                        nbonded -> 20, ntrials -> 20, niterations -> 1000}};
```

5.1.4 Examples

An example is given by

```
build           = {
  system        -> {
    id          -> main,
    density     -> 0.7,
    temperature -> 1
  },
  select        -> {
```

```
    grace       -> 0.5,
    method      -> overlap,
    theta       -> 0.3*pi,
    cluster     -> {polymer, solvent}
  }
};
```

5.2 Cancel

5.2.1 Syntax

```
cancel          = {style -> option};
```

Directive	Parameters	Description
`style`	`option`	Selects cancellation style; options are `abort` and `exit`.

5.2.2 Usage

The `cancel` command is used to stop a running script at the selected position. Its main application serves debugging purposes. Script debugging can use either `abort` or `exit`, while source code debugging would use `abort`.

5.2.3 Default

The default is given by

```
cancel          = {style -> exit};
```

5.3 Carve

5.3.1 Syntax

```
carve           = {
  mode          -> option,
  fraction      -> {real, ...},
  nsites        -> {integer, ...},
  systems       -> {constant, ...},
  inside        -> region,
  outside       -> region
};
```

Directive	Parameters	Description
`mode`	`option`	Sets mode of operation; options are `include` and `exclude`.
`fraction`	`real`	Sets the fractions in each system of the groups to be carved out; the value of fraction lies between 0.0 and 1.0; choses nsites when fraction < 0.0.
`nsites`	`integer`	Sets the number of sites remaining in the selected region in each system; number of sites in group sites increments.
`systems`	`constant`	Sets target systems; can be one or more systems.
`inside`	`region`	Sets the inner boundary of the region in which sites are to be deleted (see Section 7.7 [Region], page 196).
`outside`	`region`	Sets the outer boundary of the region in which sites are to be deleted (see Section 7.7 [Region], page 196).

5.3.2 Usage

The `carve` command carves a shape out all systems in the currently defined simulation. It will output the number of retained sites per selected system upon execution as info using the `nsites` directive when a fraction for the selected systems is defined. It will use nsites when the fraction for the selected systems is a number smaller than zero.

5.3.3 Default

The default is given by

```
carve           = {
  systems       -> 0,
  nsites        -> 0,
  mode          -> include,
  inside        -> {shape -> spheroid, type -> relative,
                    center -> {0,0,0}, radius -> {0,0,0}},
  outside       -> {shape -> spheroid, type -> relative,
                    center -> {0,0,0}, radius -> {0,0,0}}
};
```

5.3.4 Examples

An example is given by

```
variables       = {n -> 20, r1 -> 1/2, r2 -> 1/2-0.75/(2^(2/3)*n)};
carve           = {mode -> exclude,
  inside        -> {shape -> spheroid,
                    center -> {0, 0, 0}, radius -> {r2, r2, r2}},
  outside       -> {shape -> spheroid,
                    center -> {0, 0, 0}, radius -> {r1, r1, r1}}
};
```

5.4 Clusters

5.4.1 Syntax

```
clusters        = {
  cluster       -> {id -> constant, system -> constant, n -> integer,
                      [group -> constant | template -> constant]},
  ...
  polymer       -> {id -> constant, system -> constant, n -> integer,
                      type -> option, groups -> {constant, ...},
                      nrepeat -> {integer, ...}},
  ...
  graft         -> {core -> constant, type -> constant, polymer -> constant,
                      connect -> option},
  ...
  body          -> {id -> constant, system -> constant, n -> integer,
                      density -> real, [surface -> constant,]
                      core -> constant, region -> region},
  ...
};
```

Directive	Parameters	Description
`cluster`	`struct`	Defines multiple single group clusters.
`polymer`	`struct`	Defines multiple repeating group clusters.
`graft`	`struct`	Defines graft connection to a backbone cluster group.
`body`	`struct`	Defines multiple bodies constrained by a region; particles consist either one core with multiple surface sites or core sites only.
`id`	`constant`	Sets identifier, either textual or numerical.
`system`	`constant`	Sets destination system to construct in,
`n`	`integer`	Sets the number of molecules to create.
`group`	`constant`	Sets the contributing group; must be a group without branches for cluster.
`template`	`constant`	Sets the contributing template as predefined in force field files (see Section 5.11 [Field], page 106).
`nrepeat`	`integer`	Sets the number of repeat units of each contributing monomer in the polymer; one nrepeat can be given in case of multiple groups, which results in assignment of this one nrepeat to all contributing groups.
`groups`	`constant`	Sets contributing groups; must be branched and connected groups; groups will be terminated with their respective terminators as defined by the `groups` command (see Section 5.17 [Groups], page 117).

`type`	`option`	Sets the type of polymer; options are `block` for block copolymers, `alternate` for alternating copolymers, and `random` statistical copolymers.
`core`	`constant`	Sets the core cluster; must be a cluster without branches and only one contributing site; density describes core density when no surface groups are defined.
`type`	`constant`	Sets the grafting type from which to grow branch; only one grafting type per grafting core is allowed.
`connect`	`option`	Sets the growth starting point; options are `head` or `tail`.
`surface`	`constant`	Sets the surface group; must be a group without branches and only one contributing site; one core particle is assumed; density describes surface density.
`region`	`region`	Sets the region of the particle (see Section 7.7 [Region], page 196).

5.4.2 Usage

The `clusters` command creates clusters according to the desired schemes as outlined above, which populate the desired system as set by the `system` directive.

5.4.3 Default

The default is given by

```
clusters        = {
  cluster       -> {id -> 0, system -> 0, n -> 0},
  polymer       -> {id -> 0, system -> 0, n -> 0},
  body          -> {id -> 0, system -> 0, n -> 0}
};
```

5.4.4 Examples

Direct definition of molecules uses the keyword `cluster`, e.g. a configuration with 1000 water molecules is obtained through

```
groups          = {
  group         -> {id -> water, chemistry -> "O"}
};
clusters        = {
  cluster       -> {id -> water, system -> 0, n -> 1000, group -> water}
};
```

Polymers can either display a block, alternating, or random constallation of its constituing monomers. Assume three monomers, combined with one terminator,

```
groups          = {
  group         -> {
    id          -> A,
    chemistry   -> "*A*",
    connects    -> {
```

```
      {head, {A, tail}}, {head, {B, tail}}, {head, {C, tail}},
      {tail, {A, head}}, {tail, {B, head}}, {tail, {C, head}},
      {head, {T, head}}, {tail, {T, head}}
    }
  },
  group         -> {
    id          -> B,
    chemistry   -> "*B*",
    connects    -> {
      {head, {A, tail}}, {head, {B, tail}}, {head, {C, tail}},
      {tail, {A, head}}, {tail, {B, head}}, {tail, {C, head}},
      {head, {T, head}}, {tail, {T, head}}
    }
  },
  group         -> {
    id          -> C,
    chemistry   -> "*C*",
    connects    -> {
      {head, {A, tail}}, {head, {B, tail}}, {head, {C, tail}},
      {tail, {A, head}}, {tail, {B, head}}, {tail, {C, head}},
      {head, {T, head}}, {tail, {T, head}}
    }
  },
  group         -> {
    id          -> T,
    chemistry   -> "*T",
    connects    -> {
      {head, {A, head}}, {tail, {B, head}}, {tail, {C, head}},
      {head, {A, tail}}, {head, {B, tail}}, {head, {C, tail}}
    }
  }
};
```

One hundred homopolymer molecules consisting of 10 A monomers are defined by

```
clusters        = {
  polymer       -> {id -> polymer, system -> main,
                      n -> 100, nrepeat -> 10, type -> random, groups -> A}
};
```

which generates 100 clusters with TAAAAAAAAAAT as a resulting polymer sequence. Alternatively, these monomers can be combined into three types of copolymers, i.e. random, block, and alternating coplymers. A random copolymer consisting of 10 A monomers, 20 B monomers, and 30 C monomers, is defined by

```
clusters        = {
  polymer       -> {id -> polymer, system -> main,
                      n -> 100, nrepeat -> {10, 20, 30}, type -> random,
                      groups -> {A, B, C}}
};
```

in which all monomers are randomly distributed over the polymer backbone. Construction of a block copolymer consisting of 2 A, 3 B, and 4 C monomers, is obtained through

```
clusters        = {
  polymer       -> {id -> polymer, system -> main,
                      n -> 100, nrepeat -> {2, 3, 4}, type -> block,
                      groups -> {A, B, C}}
};
```

which generates TAABBBCCCCT as a resulting polymer sequence. Finally, a alternating copolymer of 2 A, 3 B, and 4 C monomers is constructed through

```
clusters        = {
  polymer       -> {id -> polymer, system -> main,
                      n -> 100, nrepeat -> {2, 3, 4}, type -> alternate,
                      groups -> {A, B, C}}
};
```

which generates TABCABCBCCT as a resulting polymer sequence. Note the effect as a result of the unequal numbers when defining monomers.

5.5 Crystal

5.5.1 Syntax

```
crystal         = {n -> {x, y, z}, miller -> {h, k, l}, plane -> option,
                    periodic -> vector};
```

Directive	Parameters	Description
`n`	`{x, y, z}`	Sets the number of crystal repeat units in `x`, `y`, `z` integer increments.
`miller`	`{h, k, l}`	Sets miller crystal plane indices specifying the desired tilt, where `h`, `k`, and `l` are integers.
`plane`	`option`	Specifies crystalline base plane; this plane is the plane which creates a crystalline cut; options are `a`, `b`, and `c`, or `x`, `y`, and `z`, or `m100`, `m010`, and `m001` respectively.
`periodic`	`vector`	Specifies wether to leave or cut connections through periodic boundaries, where vector elements `x`, `y`, and `z` are `booleans`, for which `true` leaves and `false` cuts connections in the specified direction.

5.5.2 Usage

The `crystal` command is used to build crystal copies of crystalline unit cells. These unit cells can either contain unconnected or connected molecules. Miller indices in combination with a base plane definition creates the possibility to rotate the final cell to any desired orientation using integer indices `h`, `k`, and `l`. These indices are defined using the usual conventions. Rotation allows for cleaving a crystal by a desired plane, e.g. `miller -> {2, 0, 1}` in combination with `plane -> m001` uses the plane defined by crystal base vectors $\vec{a}$ and $\vec{b}$ and rotates the cell to satisfy the miller index vector {2, 0, 1}. Box dimensions are altered to form a new unit cell which consequently can be used again as a crystalline building block. The `periodic` option allows for cutting connections (bonds) through periodic boundaries, which enables the creation of a free structure when using the `carve` command (see Section 5.3 [Carve], page 91). All vector elements are set to the given input when only one element is set, i.e. `periodic -> true` equals `periodic -> {true, true, true}`.

5.5.3 Default

The default is given by

```
crystal         = {
  plane         -> m100,
  n             -> {1, 1, 1},
  miller        -> {1, 0, 0},
  periodic      -> {false, false, false}
};
```

5.5.4 Examples

A few examples are

```
crystal         = {
```

```
  plane         -> m001,
  n             -> {5, 4, 9},
  miller        -> {0, 0, 1},
  periodic      -> true
};
crystal         = {
  n             -> {n, n, n},
  miller        -> {0, 0, 1},
  plane         -> m001,
  periodic      -> {true, true, false}
};
```

5.6 Cut

5.6.1 Syntax

```
cut             = {n -> integer, mode -> option, nmin -> integer,
                   region -> region};
```

Directive	Parameters	Description
`n`	`integer`	Defines the number clusters to be cut in each active system.
`mode`	`option`	Selects cutting mode; currently the only option is `random`.
`nmin`	`integer`	Specifies the minimum allowable number of mobile repeat units that remain.
`region`	`region`	Specifies in which region in the simulation cell to cut (see Section 7.7 [Region], page 196).

5.6.2 Usage

The `cut` command finds its main application in cutting a set of crystalline clusters during the preparation of semi-crystalline structures. It is, however, not restricted to crystalline structures alone. The `cut` command cuts exactly one bond on the specified number of clusters that fall within the specified region. The cutting mode governs the choice of affected clusters.

5.6.3 Default

The default is given by

```
cut             = {
  n             -> 0,
  nmin          -> 0,
  mode          -> random,
  region        -> {shape -> spheroid, type -> relative,
                    center -> {0,0,0}, radius -> {0,0,0}}
};
```

5.6.4 Examples

An example is given by

```
cut             = {
  n             -> 20,
  nmin          -> 3,
  region        -> {shape -> cuboid, radius -> {1/2,1/2,1/4}}
};
```

5.7 Deform

5.7.1 Syntax

```
deform          = {mode -> constant, type -> constant, system -> constant,
                       frequency -> long, density -> real, geometry -> voigt};
```

Directive	Parameters	Description
`mode`	`constant`	Sets the deform mode; options are either `none` or `affine` for the site deformation mode.
`type`	`constant`	Sets the type of geometry; options are either `relative` for reduced units or `absolute` for absolute units.
`system`	`constant`	Sets identity of system to be deformed; can be either textual or numerical.
`frequency`	`long`	Sets the frequency with which to perform incremental deformations; can either be set to `instant`, forcing instantaneous deformation, or to a positive integer, expressing the step size (in system cycles) in which deformation is gradually applied over the course of a run.
`density`	`real`	Sets the final system density; can either be set to `calculate` when chosing or forced when chosing values > 0.0.
`geometry`	`voigt`	Sets the final system geometry; used as a fractional size when `density` is set, otherwise used as system dimensions from which a density is calculated (see Section 7.15 [Voigt], page 367).

5.7.2 Usage

The `deform` command is used to gradually deform a selected `system` to the desired target `density` over the cause of a `run` (see Section 5.29 [Run], page 139). Mobile sites are affinely deformed, rigid sites are deformed with respect of the center of mass of the rigid object, and fixed sites stay undeformed.

5.7.3 Default

The default is given by

```
deform          = {mode -> none, system -> 0, frequency -> 1,
                       density -> calculate, geometry -> {1, 1, 1}};
```

5.7.4 Examples

Examples are given by

```
deform          = {system -> main, frequency -> 2, density -> 0.7};
deform          = {system -> main, frequency -> 2, geometry -> {1, 1, 3}};
```

5.8 Delete

5.8.1 Syntax

```
delete          = {mode -> option, unwrap -> boolean, fraction -> real,
                   focus -> struct, region -> struct};
```

Directive	Parameters	Description
`mode`	`option`	Sets wether to include or exclude sites in the selected region from deletion; options are `include` and `exclude`.
`unwrap`	`boolean`	Sets wether to unwrap already existing clusters (only the first instance before a build should be set to `true`); options are `true` and `false`.
`fraction`	`real`	Defines the distribution fraction; valid values are between 0 and 1, inclusive the extremes; currently forced to be `1`.
`focus`	`struct`	Sets the selection to focus on (see see Section 7.3 [Focus], page 181).
`inside`	`struct`	Sets the inside region to consider (see see Section 7.7 [Region], page 196).
`inside`	`struct`	Sets the outside region to consider (see see Section 7.7 [Region], page 196).

5.8.2 Usage

The `delete` command is used to delete clusters at the growing sides of a simulation box. Specific selection criteria can be set by defining inside and outside regions in which the center of mass of targetted clusters need to reside. Inside and outside regions can be used to mimic a core-shell approach. A full region is considered when only the outside region is defined. Additionally, a focussing selection can be specified based on site types, group ids, and cluster ids. Undefined `focus`, `inside`, `outside` directives imply considering all active sites in the system.

5.8.3 Default

The default is given by

```
delete          = {mode -> include, unwrap -> true, fraction -> 1.0};
```

5.9 Duplicate

5.9.1 Syntax

```
duplicate       = {source -> system, destination -> {min, max}};
```

Directive	Parameters	Description
`source`	`system`	Defines the source system.
`destination`	`{start, end}`	Defines the destination systems.

5.9.2 Usage

Take source system `system` and copy it to destination starting at `min` and ending at `max`. Destination systems overlapping with the source system is not allowed.

5.10 Export

5.10.1 Syntax

```
export          = {
  profile       -> {name -> string, type -> option, style -> option},
  ...
  sample        -> {name -> string, type -> option, style -> option},
  ...
};
```

Directive	Parameters	Description
profile	struct	Export settings for profiles.
sample	struct	Export settings for sampled data.
name	string	Defines the file name containing exported data.
type	option	Indicates the type of profile or sample to export; available profiles are angle, bond, bridge, density, force, loop, mass, order, and tail; the available sample is cavity.
style	option	Export format; options are csv, json, and math for comma separated value, JSON, and Mathematica formats respectively.

5.10.2 Usage

The export command is used to export data resulting from structure analysis into individual files, using mathematica or comma seperated value formats. Only distributions resulting from analyses are exported.

5.10.3 Default

The export command does not have any defaults.

5.10.4 Examples

Examples of the export command are

```
  export        = {
    profile     -> {name -> "density", type -> density, style -> math},
    profile     -> {name -> "bond", type -> bond, style -> csv},
    sample      -> {name -> "cavity", type -> cavity, style -> math}
  };
```

5.10.5 Data Interpretation

export a distribution by using the export command, e.g.

```
  export        = {name -> "bridge", type -> bridge, style -> csv};
```

which exports the measured tail distribution to file name bridge.csv in comma separated format. An alternate style is math, which exports to bridge.m in Mathematica format. EMC output format is equivalent to Mathematica format. I typically use the latter, since it allows me to create notebooks for evaluation, which can be used within workflows.

For example, the bridge distribution under the profiles header in the output file contains of a header, followed by a distributions block:

```
profiles = {
  bridge -> {
    active -> true, convolute -> false, direction -> 0,
    nlevels -> 1, binsize -> 1, ndistributions -> 1, distributions -> {
      ncalls -> 101, offset -> 7, nbins -> 645, data -> {
        {n -> 101, accu -> 101, weight -> 101},
        ...
      }
    }
  }
};
```

The header contains general distribution information, i.e. if the distribution is actively sampled, whether to convolute the data (when applicable), in which direction was sampled, how many levels it has, the binsize, and the number of distributions (one for each system). Levels are used to allow for 3D density sampling.

The distributions also contain a header section, which is followed by a data section. The header records for each distribution how many calls a distribution has, what offset the data is recorded at and how many bins the distribution contains. One can construct the x coordinates by combination of the offset with the bin size and the number of bins, i.e.

$x_i = (i + offset)binsize$, with $0 \leq i < nbins$.

The data section entries contain the y coordinates in three contributions: the frequency of calls to this bin, the accumulated scalar data, and the weight with which this data data was recorded. These three numbers are always the same for histograms (e.g. bridge, loop and tail distributions) and represent the frequency of a certain length or size, i.e.

$y_i = n$

For order profiles and other slab-wise averages, the accumulated data needs to be divided by their corresponding weight in order to get a local density or order parameter respectively, i.e.

$y_i = accu/weight$

For density profiles, the accumulated data needs to be divided by the number of distribution calls (ncalls) in order to get the local density, i.e.

$$y_i = accu/ncalls$$

Interpretation of cavity distributions is identical as for any histogram distribution. See the above description for brigde distributions.

5.11 Field

5.11.1 Syntax

```
field           = {
  id            -> identifier,
  mode          -> option,
  name          -> {string, ...},
  compress      -> boolean,
  apply         -> boolean,
  error         -> boolean,
  debug         -> boolean,
  angle         -> option,
  torsion       -> option,
  improper      -> option,
  increment     -> option,
  allow         -> option
};
```

Directive	Parameters	Description
`id`	`constant`	Sets the subfield identifier, e.g. used in groups to identify the field relevant for typing.
`mode`	`option`	Sets execution, input, output, or import mode; EMC standard options are `apply`, `get`, and `put`; options `cff`, `charmm`, `dpd`, `martini`, and `opls` import native and non-native force field formats.
`name`	`string`	Sets file names of either input or output; all import options expect a general parameter file followed by an optional topology file.
`compress`	`boolean`	Sets compression flag; options are `true` or `false`.
`apply`	`boolean`	Apply force field typing directly after reading; options are `true` or `false`.
`error`	`boolean`	Sets error flag; options are `true` or `false`; exits on error when true.
`debug`	`boolean`	Set debug flag, which controls the output of rule interpretation debugging information; options are `true` or `false`.
`angle`	`option`	Sets angle field flag; options are `ignore`, `complete`, `warn`, `empty`, or `error` (see see Section 7.2 [Field Flags], page 180).
`torsion`	`option`	Sets torsion field flag; options are `ignore`, `complete`, `warn`, `empty`, or `error` (see see Section 7.2 [Field Flags], page 180).
`improper`	`option`	Set improper field flag; options are `ignore`, `complete`, `warn`, `empty`, or `error` (see see Section 7.2 [Field Flags], page 180).

`increment`	`option`	Set increment field flag; options are `ignore`, `complete`, `warn`, `empty`, or `error` (see see Section 7.2 [Field Flags], page 180).
`allow`	`option`	Sets which selection is considered during typing; options are `all` or `template`.

5.11.2 Usage

The `field` command allows for force field typing. A native format governs storage through `get` and `put` modes. The `apply` mode executes force field typing and should be invoked after `clusters` assignment only. Currently `field` supports the Accelrys BIOSYM force field format (`mode -> cff`) for importing force field typing rules (`.dat` files) and parameter assignment tables (`.frc` files). CHARMM, DPD, MARTINI, and OPLS force fields (`mode -> charmm`, `dpd`, `martini`, or `opls` respectively) are imported using an EMC native textual format (`.prm` and `.top` files). A conversion script for CHARMM force fields is given by `./scripts/charmm.pl`, for MARTINI force fields by `./scripts/martini.pl`, and for OPLS force fields by `./scripts/opls.pl`. Force fields can contain templates, which are used to define groups (see Section 5.17 [Groups], page 117) and clusters (see Section 5.4 [Clusters], page 93). Force fields are found in `./field` and subsequent directories. Note, that each force field only includes chemistries, for which it was parameterized. This means, that not all chemistries can be typed by all force fields. Also be aware, that typing is not guaranteed to be correct. It is kindly requested to report bugs, so that improvements to force fields can be included in next versions.

Internally, force fields consist of a set of general and specific parameters. General parameters contain automatic parameters, which are used when specific parameters are not defined. Aside from parameters, templates can be defined for commonly used chemistries. Martini force fields use templates to preset the parametrized substances. Note, that the abovementioned Martini script can add types when doubles occur. Here, the use of templates is advised to avoid the selection of wrong types.

5.11.3 Formats

Modes `get` and `put` support an EMC-native textual format, which encompasses most possibilities, as needed by supported force fields. Support of leaner textual formats is provided as short hand for ease of use.

The CFF file format is identical to the `.frc` and `.dat` as provided by Accelrys' Materials Studio. It has been tested for PCFF and COMPASS formats.

The CHARMM, DPD, MARTINI, and OPLS file formats are created for EMC and function as intermediates between CHARMM, DPD, GROMACS, or OPLS force field families and EMC. Data blocks in the EMC .prm format are flanked by the keywords `ITEM` and `ITEM END`. The keyword `ITEM` is followed by an identifier to designate the following data block. This data block describes data in tab-separated columns. Note, that only exactly one tab is allowed between entries. Spaces and multiple tabs lead to erroneous behavior. The interpretation of the data depends on the `ITEM` identifier as described in the following table:

ITEM ID	**Parameters**	**Description**
`DEFINE`	See next table	Contains force field definitions and units.

MASS	type mass element nconnections charge comment	Defines masses; columns contain a textual type, a numerical mass value, a textual element, a numerical number of connections, a numerical charge, and a textual comment line respectively.
RULES	id type element {index|residue atom} charge rule	Defines typing rules; columns contain a numerical id, a textual type, a textual element, a numerical index (e.g. OPLS index), a numerical charge, and a rule in SMILES format respectively; in case of the CHARMM force field, the single numerical index entry is replaced by two entries for residue and atom, respectively.
COMMENTS	id index element comment	Defines comments associated with rules; columns contain a numerical id, a numerical index (e.g. OPLS index), a textual element, and a textual comment respectively.
LITERATURE	year volume page journal	Lists literature references associated with force field; columns contain a numerical year, a textual cvolume, a textual page, and a textual journal reference respectively.
EQUIVALENCE	type pair bond angle torsion improper	Tabulates equivalences between types (used to reference a derived type to its original; pair types are considered unique); all columns are in a textual format.
TEMPLATE	name smiles	Lists predefined group and cluster templates; all columns are in a textual format; SMILES format follows the Daylight Chemical Information Systems definition.
PRECEDENCE		Precendences describe the order in which rules are interpreted; parenthesis indicate precedence hierarchy: (? (c (c1)(c2))) c followed by c1 and c2.
NONBOND	type1 type2 sigma epsilon	Defines the interaction parameters for Lennard-Jones 6-12 pairwise interactions; columns contain two textual types followed by two numerical values.
BOND, BOND_AUTO	type1 type2 k l0	Defines covalent bond length interaction parameters for a harmonic spring; columns contain two textual types followed by two numerical values.
ANGLE, ANGLE_AUTO	type1 type2 type3 k theta0	Defines bond angle interaction parameters for a harmonic spring; columns contain three textual types followed by two numerical values.
TORSION, TORSION_ AUTO	type1 type2 type3 type4 k n delta [...] [index]	Defines bond torsion interaction parameters for a fourier expansion; columns contain four textual types followed by up to four groups of three numerical values; the last column can contain an optional OPLS reference index.
IMPROPER, IMPROPER_ AUTO	type1 type2 type3 type4 k psi0	Defines improper interaction parameters; columns contain four textual types followed by two numerical values; each of the three possible angle combinations contribute 1/3 to the total improper energy.

Valid DEFINE IDs are given in the following table:

DEFINE ID	Type	Description
FFNAME	string	Defines force field name.
FFTYPE	string	Defines force field type; options are ATOMISTIC, UNITED, or COARSE.
VERSION	string	Defines force field version.
CREATED	string	Defines parameter file creation date.
LENGTH	string	Sets length unit of parameters in file; options are ANGSTROM, NANOMETER, MICROMETER, METER, or REDUCED.
ENERGY	string	Sets energy unit of parameters in file; options are J/MOL, KJ/MOL, CAL/MOL, KCAL/MOL, or REDUCED.
DENSITY	string	Sets density units of parameters in file; options are G/CC, KG/M^3, or REDUCED.
MIX	string	Sets nonbonded mixing rule; options are NONE, BERTHELOT, ARTIHMETIC, GEOMETRIC, or SIXTH_POWER.
NBONDED	integer	Sets the number of bonds excluded from nonbonded pairwise calculations.
ANGLE	string	Include angle considerations during typing; options are IGNORE, COMPLETE, WARN, EMPTY or ERROR.
TORSION	string	Include torsion considerations during typing; same options as for ANLGE.
IMPROP	string	Include improper considerations during typing; same options as for ANLGE.

The distinction between general and specific force field entries is made by adding extension _AUTO to a parameter keyword. Allowed keywords for this extension are PRECEDENCE, BOND, ANGLE, TORSION, and IMPROPER.

A conversion script for the OPLS force field is provided by ./scirpts/opls.pl, which creates a .prm and .top from OPLS input formats .par and .sb. It is used through

```
opls.pl [-source=source] target
```

The script assumes the existence of source.par, source.sb, and target.define. The latter is an extra file providing typing rules and possible additions or redefinitions of the OPLS source. The .define files found in ./field/opls/2012/ are currently under development.

A conversion script for the MARTINI force field in GROMACS format is provided by ./scripts/martini.pl. The script is called separately for each contribution through

```
martini.pl name[.itp]
```

The use of multiple topology files within one simulation can be obtained by using a master topology file, which consists of a concatenation of individual topology files. The martini.pl script will create extra types in case of redefinition of bonds, angles, torsions, or impropers.

MARTINI files can be found in `./field/martini/v2.0`. A converted copy of the nonbonded parameter file `martini.itp` is stored as `martini.prm` in this directory.

A conversion script for the CHARMM force field is provided by `./scirpts/charmm.pl`, which creates a `.prm` and `.top` from CHARMM input formats `.prm` and `.rtf`. It is used through

```
charmm.pl [-source=source] target
```

The script assumes the existence of `par_source.prm`, `top_source.rtf`, and `target.define`. The latter is an extra file providing typing rules and possible additions or redefinitions of the OPLS source. A conversion of `all27_prot_lipid` with its accompanying `.define` file as can be found in `./field/charmm/c32b1/`. The CHARMM force field interpretation is still in beta phase.

5.11.4 Default

Unless otherwise stated, the default is given by

```
field   = {
  mode  -> get, compress -> false, error -> true,
  angle -> true, torsion -> true
};
```

5.12 Flag

5.12.1 Syntax

```
flag            = {system -> id, oper -> mode, select -> mode,
                   flag -> {constant, ...},
                   site -> {constant, ...},
                   group -> {constant, ...},
                   cluster -> {constant, ...}, region -> region};
```

Directive	Parameters	Description
`system`	`id`	Sets system to operate on; identifiers as previously defined.
`oper`	`mode`	Sets how to operate on site flag; modes are `set` or `unset`.
`select`	`mode`	Sets selection (used for ignoring head-tail (i.e. ends) connections in crystal builds); modes are `all` or `ends`.
`flag`	`constant`	Sets site flag; input either one or more flags; options are `fixed`, `semi`, `rigid`, or `branch`.
`site`	`constant`	Sets the site types to include; all site types are implied when left blank.
`group`	`constant`	Sets the group to include; all groups are implied when left blank.
`cluster`	`constant`	Sets the cluster id to include; all cluster ids are implied when left blank.
`region`	`region`	Specifies in which region in the simulation cell to cut (see Section 7.7 [Region], page 196).

5.12.2 Usage

The `flag` command is used to define site-based functionalities through flags. Sites store these flags in a bitwise representation. Possible flag options are given in the table below.

Name	Bit	Description
`fixed`	`0b0000000001`	Prevents the site from being included in energy calculations and from being moved during simulation.
`semi`	`0b0000000010`	Prevents the site from being moved during simulation, however, it is included in energy calculations.
`head`	`0b0000000100`	Defines a site as a head site and is normally not set in a global fashion, but only on a per site basis (as used in data files).
`tail`	`0b0000001000`	Defines a site as a tail site and is normally not set in a global fashion, but only on a per site basis (as used in data files).
`rigid`	`0b0000010000`	Defines a site as part of a rigid body and causes a collection of sites to move as one object; only connected sites end up in one rigid body when the `body` flag is not set.

`body`	`0b0000100000`	Defines a site as being part of a body as defined by the region on the cluster as to which the site belongs; sites will either move inside or on the surface of this body; one cluster represents one body; all rigid sites within this body are treated as one rigid body.
`surface`	`0b0001000000`	Defines a site to be on the surface of a body rather than in its interior.
`backbone`	`0b0010000000`	Defines a site to be part of a cluster backbone; backbones are determined internally upon initialization and are able to take part in topology altering moves (e.g. endbridge and reptate moves).
`branch`	`0b0100000000`	Defines a site to be part of a cluster branch; branches are excluded from topology altering moves (e.g. endbridge and reptate moves).
`unbound`	`0b1000000000`	Defines a site to be part of a cluster branch; branches are excluded from topology altering moves (e.g. endbridge and reptate moves).

5.12.3 Default

The default is given by

```
flag            = {system -> 0, oper -> set, select -> all,
                   region -> {
                     shape -> cuboid, type -> relative,
                     center -> {0, 0, 0}, radius -> {1, 1, 1}};
```

5.13 Focus (Command)

5.13.1 Syntax

```
focus           = {
  sites         -> {constant, ...},
  groups        -> {constant, ...},
  clusters      -> {constant, ...}
};
```

Directive	Parameters	Description
`sites`	`constant`	Sets site or sites to focus on; refers to mass type constants.
`groups`	`constant`	Sets group or groups to focus on.
`clusters`	`constant`	Sets cluster or clusters to focus on.

5.13.2 Usage

The `focus` command performs a translation to the origin of the accumulative center of mass of all participating sites, groups, and/or clusters. Sites, groups, and clusters are defined by constants as referred to in the simulation constant definition paragraph (see Section 7.1 [Constants], page 179) and described in the focus section (see Section 7.3 [Focus], page 181).

5.13.3 Default

Unless otherwise stated, the default is given by

```
focus   = {};
```

5.14 Force

5.14.1 Syntax

```
force           = {style -> option, message -> boolean};
```

Directive	Parameters	Description
`style`	`option`	Selects force style; valid options are `none`, `init`, or `list`
`message`	`option`	Selects output to screen; alid options are `none`, `raw`, `nkt`, or `n`, which corresponds to energy in no particular units, internal units, units of nkT or units of n, respectively; internal units correspond to the units of the selected force field (e.g. kcal/mol for COMPASS, CHARMM, OPLS, GROMACS, or MARTINI).

5.14.2 Usage

The `force` command is used to calculate and possibly output energies and virials associated with all systems in the active simulation. It allows for an internal check of consistency of energies after a simulation by calculation of energies either directly by using the `none` option, or using the neighbor list resulting from a simulation using the `list`, or through reinitialization of the neighbor lists by using the `init` option. The energy output is controlled by the `message` directive. Energies and virials are reported in either units as defined in the units paragraph when selecting `raw`, or normalized by the number of active sites when selecting `n`, or in relative units of nk_BT when selecting `nkt`.

5.14.3 Default

The default is given by

```
force           = {style -> init, message -> nkt};
```

5.15 Former

5.15.1 Syntax

```
former          = {name -> string, mode -> option, forcefield -> option};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .emc is assumed.
`mode`	`option`	Selects i/o mode; the only currently supported option is `get`.
`forcefield`	`option`	Selects the desired force field; valid options are `none`, `standard`, and `charmm`.

5.15.2 Usage

The `former` command is used to import data files generated by previous EMC versions. Currently only the import of version 8 data files is supported.

5.15.3 Default

The default is given by

```
former          = {mode -> get, forcefield -> none};
```

5.16 Get

5.16.1 Syntax

```
get             = {name -> string, compress -> boolean};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.

5.16.2 Usage

The `get` command is used to import data files generated by the current version of EMC, using a structured data format. Data files function as a restart file and are expected to be in a textual format.

5.16.3 Default

The default is given by

```
get             = {compress -> false};
```

5.17 Groups

5.17.1 Syntax

```
groups          = {
  group         -> {
    id          -> constant,
    charge      -> real,
    charges     -> option,
    template    -> constant,
    terminator  -> boolean,
    chemistry   -> string,
    depth       -> integer,
    field       -> constant,
    connects    -> {{source -> integer,
                     destination -> {index -> constant, site -> integer}
                    }, ...}
  },
  delete        -> {
    id          -> constant,
    site        -> integer
  },
  ...
};
```

Directive	Parameters	Description
`group`	`struct`	Defines group parameters; repeating structure.
`id`	`constant`	Sets group identity; can be either numerical or textual.
`charge`	`double`	Sets an additional charge of the complete group, which value divided by the number of group sites is added to the site charges originating from force field or override charge assignments.
`charges`	`option`	Defines how partial charges defined in SMILES string are dealt with; valid options are `forcefield` for using partial charges assigned by a force field only (see Section 5.11 [Field], page 106), `additive` for adding specified partial charges per SMILES atom to values resulting from force field assigment, and `override` for only using the specified partial charges (non-specified charges are zero).
`template`	`constant`	Selects a template as predefined in force field files; can be either numerical or textual (see Section 5.11 [Field], page 106); templates replace the use of a chemistry string.
`terminator`	`boolean`	Indicates if this group can be used as a terminator, which is useful for coarse-grained polymers without termination groups; options are `true` or `false`.

`chemistry`	`string`	Sets the group's chemistry using the SMILES format (see Section 7.8 [SMILES], page 197).
`depth`	`integer`	Sets recursive depth for ring determination during SMILES interpretation; the depth should equal the maximum number of atoms participating in a ring within the given SMILES string; standard option is `auto`; any set depth smaller than 3 reverts to the `auto` setting.
`field`	`constant`	Optionally set a specific subfield identifier to use when typing this group (see Section 5.11 [Field], page 106).
`connects`	`struct`	Defines connections to other groups; both `source` and `site` correspond to the position in the SMILES of where a connection occurs; '`*`', '`>`', or '`<`' characters mark connection points; shortcuts `head`, `tail`, and `$end1` through `$end10` denote connections in a group withouth having to state their exact positions within the corresponding SMILES; `head` and `tail` are analogous to `$end1` and `$end2` respectively (see example below).
`source`	`integer`	Sets the source element
`destination`	`struct`	Defines the destination group and element
`index`	`constants`	Sets the destination group; can be either numerical or textual
`site`	`integer`	Sets the destination element
`delete`	`struct`	Deletes a specified group site corresponding to a chemistry entry; all corresponding simulation sites are deleted accordingly.
`id`	`constant`	Target group identity; can be either numerical or textual.
`site`	`integer`	Site number corresponding to the entry position in the SMILES string; counting starts at 0.

5.17.2 Usage

The `groups` command creates the necessary groups used to identify full molecules or chemical repeat units within clusters, as represented by SMILES strings. Keyword `charges` is used to assign extra charge to a subset of atoms in case of ionic liquids, for which case a force field might not supply charged molecules. One can also use keyword `charge` to increase or decrease the charge of a full group. This option assigns a partial charge to each site, which is equal to the full charge divided by the number of sites in the group.

5.17.3 Default

The default is given by

```
groups          = {
  group         -> {
    id          -> 0,
    charge      -> 0,
    charges     -> forcefield,
```

```
    terminator  -> false
  }
};
```

5.17.4 Examples

A few atomistic examples are

```
groups          = {
  group         -> {
    id          -> water,
    chemistry   -> "HOH"
  },
  group         -> {
    id          -> dodecane,
    chemistry   -> "(C)12"
  },
  group         -> {
    id          -> benzene,
    chemistry   -> "c1ccccc1"
  },
  group         -> {
    id          -> acid,
    chemistry   -> "C(=O)[O-]
  }
};
```

An example of deleting a group site is given by

```
groups          = {
  delete        -> {
    id          -> dodecane,
    site        -> 0
  }
};
```

which deletes the first site of group dodecane, effectively changing dodecane into undecane. Note, that all corresponding simulation sites are deleted accordingly.

An example of a set of monomers for a coarse-grained polymer is given by

```
groups          = {
  group         -> {
    id          -> monomer1,
    chemistry   -> "*ab*c(d)cc",
    connects    -> {
      {head, {monomer1, tail}}, {head, {monomer2, head}},
      {tail, {monomer1, head}}, {tail, {monomer2, head}},
      {head, {terminator, head}}, {tail, {terminator, head}}}
  },
  group         -> {
```

```
    id          -> monomer2,
    chemistry   -> "*c*",
    connects    -> {
      {head, {monomer1, head}}, {head, {monomer1, tail}},
      {head, {monomer2, head}}, {head, {monomer2, tail}},
      {head, {terminator, head}}}
  },
  group         -> {
    id          -> terminator,
    chemistry   -> "*t",
    connects    -> {
      {head, {monomer1, head}}, {head, {monomer2, head}}}
  },
};
```

The '*' character implies the start and end or head and tail of a repeat unit of a polymer. By default, a group is not a terminator when more than one '*' character appears in the SMILES string. However, when such a group should also function as a terminator, the `terminator` flag can be used as indication of such. A group is automatically a terminator when a '*' character appears only once, which means, that the `terminator` flag need not be set. The keyword `connects` lists the possible connections of these start and end sites to sites in the same or other groups. The keywords `head` and `tail` represent internal short cuts and indicate the position of the '>' and '<' characters repectively.

5.18 Insight

5.18.1 Syntax

```
insight         = {name -> string, compress -> boolean,
                   system -> integer, mode -> option,
                   forcefield -> option, atomistic -> option,
                   charges -> boolean, formal -> boolean, depth -> integer,
                   detect -> boolean, cut -> boolean, pbc -> boolean,
                   map -> boolean, unwrap -> boolean, percolate -> boolean,
                   crystal -> boolean, flag -> struct};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.
`system`	`integer`	Selects system to be imported or exported
`mode`	`option`	Selects i/o mode; supported options are `get` or `put`.
`forcefield`	`option`	Selects the desired force field for writing; see Section 7.5 [Ports], page 194, for valid options; option `auto` auto-selects.
`atomistic`	`option`	Select the atomistic mode normally defined by the choice of `forcefield`; options are `none`, `united`, or `full`.
`charges`	`boolean`	Use partial charges from imported structures; options are `true` or `false`.
`formal`	`boolean`	Use formal charges from imported structures; options are `true` or `false`.
`detect`	`boolean`	Detect and assign atom and amino acid residue types; options are `true` or `false`.
`depth`	`integer`	Sets recursive depth for ring determination during SMILES interpretation; the depth should equal the maximum number of atoms participating in a ring within the given SMILES string; standard option is `auto`; any set depth smaller than 3 reverts to the `auto` setting.
`cut`	`boolean`	Selects cut mode: cut bonds that cross cell boundaries; options are `true` or `false`.
`pbc`	`boolean`	Applies periodic boundary conditions; options are `true` or `false`.
`map`	`boolean`	Selects mapping sites and box geometry according to a minimum image convention; options are `true` or `false`.
`unwrap`	`boolean`	Unwraps clusters to cross periodic boundaries with application of periodic boundary conditions to its center of mass when applicable; options are `true` or `false`.

`crystal`	`boolean`	Imported structure is crystalline; options are `true` or `false`.
`percolate`	`boolean`	Allow for percolating structures; options are `true` or `false`.
`flag`	`struct`	Allows for setting system flags while importing morphologies; see see Section 7.10 [System Flags], page 201, for the various available flags.

5.18.2 Usage

The `insight` command is used to import and export InsightII data files in CAR and MDF format for exchange of configurations between EMC and Materials Studio, a product of Accelrys[1].

5.18.3 Default

The default is given by

```
insight         = {compress -> false,
                   system -> 0, forcefield -> auto, detect -> false,
                   cut -> false, pbc -> true, map -> false, unwrap -> true};
```

5.18.4 References

1. Materials Studio, '`http://www.accelrys.com/`'

5.19 Lammps

5.19.1 Syntax

```
lammps          = {name -> string, compress -> boolean,
                   system -> integer, mode -> option,
                   units -> option, length -> double, forcefield -> option,
                   shake -> option, atomistic -> boolean, cutoff -> boolean,
                   charges -> boolean, ewald -> boolean,
                   bonds -> boolean, types -> boolean,
                   parameters -> boolean, cross -> boolean,
                   variables -> boolean, coefficients -> boolean,
                   comment -> boolean, map -> boolean, unwrap -> boolean,
                   version -> integer, flag -> structure};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .data is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.
`system`	`integer`	Selects system to be imported or exported
`mode`	`option`	Selects i/o mode; supported options are `get` or `put`.
`length`	`double`	Defines a scaling length to superceede the simulation variable `units -> angstrom`; the latter is used when length is not specified (see Section 7.13 [Units], page 364).
`units`	`option`	Selects wether to represent site coordinates in reduced or real units, for which options are `lj` or `real` respectively; alternatively `none` is also defined.
`forcefield`	`option`	Selects the desired force field; see Section 7.5 [Ports], page 194, for valid options.
`shake`	`option`	Selects the SHAKE[1,2] mode for atomistic force fields; valid options are `none`, `auto`, `hydrogen`, `water` and `all`; `auto` bases the choice on the selected force field (`none` by default).
`atomistic`	`boolean`	Selects atomistic mode: no charge column and no bonds; options are `true` or `false`.
`cutoff`	`boolean`	adds a column containing cut offs as used for EMC force field evaluation to the LAMMPS parameter file; helpful for transferring purely repulsive potentials; options are `true` or `false`.
`charges`	`boolean`	Selects charge mode: a charge column is present in the Atoms paragraph; options are `true` or `false`.
`ewald`	`boolean`	Selects the usage of Ewald summations for long-range charge treatment; options are `true` or `false`.

`bonds`	`boolean`	Selects bonds mode: all bonded contributions are present; options are `true` or `false`.
`types`	`boolean`	Selects types mode: all types parameter definitions are present; options are `true` or `false`.
`parameters`	`boolean`	Selects the separate output of variables and (non)bonded interaction parameters in LAMMPS input script format; an extension of .params is assumed; options are `true` or `false`.
`cross`	`boolean`	Includes nonbonded cross-terms in .params file; options are `true` or `false`.
`variables`	`boolean`	Selects adding simulation variables to the separate output of variables and interaction parameters; options are `true` or `false`.
`coefficients`	`boolean`	Selects adding interaction parameters to the separate output of variables and interaction parameters; options are `true` or `false`.
`comment`	`boolean`	Includes commented references to mass names in .data and .params files; options are `true` or `false`.
`map`	`boolean`	Selects mapping sites and box geometry according to a minimum image convention; options are `true` or `false`.
`unwrap`	`boolean`	Selects unwrapping of clusters; options are `true` or `false`.
`version`	`integer`	Set LAMMPS version.
`flag`	`structure`	Set system flags.

5.19.2 Usage

The `lammps` command is used to import and export data files generated by LAMMPS, a Molecular Dynamics code conceived at Sandia National Laboratories.[3,4] Textual input formats for LAMMPS are ambiguous and are not self-defining, which creates the need for predefinition of the desired format. This port allows for commonly used force field modes and site (atom) definitions.

5.19.3 Default

The default is given by

```
lammps          = {compress -> false, system -> 0, mode -> put,
                   units -> none, forcefield -> auto, shake -> auto,
                   atomistic -> true, charges -> false, ewald -> false,
                   bonds -> false, bonds -> false, types -> true,
                   parameters -> false, cross -> false, variables -> true,
                   coefficients -> true, comment -> true, map -> true,
                   unwrap -> false};
```

5.19.4 Examples

A few examples are given by

```
lammps          = {name -> "benzene", compress -> true,
```

```
                    mode -> put, forcefield -> opls, types -> false,
                    parameters -> true, charges -> true, ewald -> true};
lammps          = {name -> "polystyrene", compress -> true,
                    mode -> put, forcefield -> coarse, types -> false,
                    parameters -> true};
lammps          = {name -> "polyethylene", mode -> put};
```

5.19.5 References

1. Ryckaert, J.-P.; Ciccotti, G. and Berendsen, H.J.C., "", *J. Comp. Phys.* **1977**, *23*, 327-341
2. Andersen, H. "", *J. Comp. Phys.* **1983**, *52*, 24-34
3. LAMMPS - Molecular Dynamics Simulator, '`http://lammps.sandia.gov/`'
4. Plimpton, S., "Fast Parallel Algorithms for Short-Range Molecular Dynamics", *J. Comput. Phys.* **1995**, *117*, 1-19

5.20 Memory

5.20.1 Syntax

```
memory          = {style -> option};
```

Directive	Parameters	Description
`style`	`option`	Sets the output style; options are `full` or `summary`.

5.20.2 Usage

The `memory` command outputs the memory consumption of the current simulation to screen. Either a summary or a full description can be selected.

5.20.3 Default

The default is given by

```
memory          = {style -> full};
```

5.20.4 Examples

An example is given by

```
memory          = {style -> summary};
```

5.21 Moves

5.21.1 Syntax

```
moves           = {
  ncycles       -> integer,
  cycle         -> integer,
  move          -> integer,
  moves...
};
```

Directive	Parameters	Description
ncycles	integer	Sets the total number of simulation cycles.
cycle	integer	Sets the current simulation cycle
move	integer	Sets the current simulation move.
moves		Access to various move settings; see Section 7.4 [Moves], page 182, for further information.

5.21.2 Usage

This variable style describes types (see Section 7.4 [Moves], page 182, for further information) and allows direct access to all variables and parameters stored within the moves structure.

5.21.3 Default

Unless otherwise stated, the default is given by

```
Unless otherwise stated, the default is given by
@verbatim
moves           = {
  ncycles       -> 0,
  cycle         -> 0,
  move          -> 0
};
```

By default, all moves are activated with zero frequency, with the exception of the displacement move (see Section 7.4.6 [Displace], page 185), which has a frequency of one.

5.22 PDB

5.22.1 Syntax

```
pdb             = {name -> string,
                   compress -> boolean, mode -> option, system -> integer,
                   length -> real, forcefield -> option, atomistic -> option,
                   charges -> boolean, depth -> integer, detect -> boolean,
                   atom -> option, residue -> option, segment -> option,
                   vdw -> option, hexadecimal -> boolean,
                   cut -> boolean, pbc -> boolean, map -> boolean,
                   unwrap -> boolean, rigid -> boolean, fixed -> boolean,
                   connectivity -> boolean, crystal -> boolean,
                   element -> option};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; unpacking and packing is added to the VMD script; options are `true` or `false`.
`mode`	`option`	Selects i/o mode; supported options are `get` and `put`
`system`	`integer`	Selects system to be imported or exported.
`length`	`real`	Set the length scale with which to scale the resulting coordinates (normmaly results from choice of `forcefield`).
`forcefield`	`option`	Selects the desired force field; see Section 7.5 [Ports], page 194, for valid options; option `auto` auto-selects.
`atomistic`	`option`	Select the atomistic mode normally defined by the choice of `forcefield`; options are `none`, `united`, or `full`.
`charges`	`boolean`	Use charges from imported structures; options are `true` or `false`.
`depth`	`integer`	Sets recursive depth for ring determination during SMILES interpretation; the depth should equal the maximum number of atoms participating in a ring within the given SMILES string; standard option is `auto`; any set depth smaller than 3 reverts to the `auto` setting.
`detect`	`boolean`	Detect and assign atom and amino acid residue types; options are `true` or `false`.
`atom`	`option`	Sets options for atom name representation upon output; options are `detect` for detection of atom types, `index` for use of EMC mass element ids, or `series` for indexing using EMC element ids followed with sequential numbering per segment. The latter is useful for CHARMM tools expecting unique atom ids.

`residue`	`option`	Sets options for residue name representation upon output; options are `detect` for detection of amino acid residues, `index` for use of EMC group ids, or `series` for increments starting with the letter R.
`segment`	`option`	Sets options for segment name representation upon output; options are `detect` for detection, `index` for use of EMC cluster ids, or `series` for increments starting with the letter M.
`vdw`	`boolean`	Include the definition for switching on Van der Waals in the resulting VMD script; options are `true` or `false`.
`hexadecimal`	`boolean`	Use hexadecimal representations for residue sequence numbers; options are `true` or `false`.
`cut`	`boolean`	Selects whether cut bonds that cross cell boundaries upon output; options are `true` or `false`.
`vdw`	`boolean`	Add Van der Waals representation to the written VMD script; options are `true` or `false`.
`pbc`	`boolean`	Applies periodic boundary conditions upon output; options are `true` or `false`.
`map`	`boolean`	Selects mapping sites and box geometry according to a minimum image convention; options are `true` or `false`.
`unwrap`	`boolean`	Unwraps clusters upon output to cross periodic boundaries with application of periodic boundary conditions to its center of mass when applicable; options are `true` or `false`.
`rigid`	`boolean`	Unwrap sites in clusters, that are flagged as rigid; options are `true` or `false`.
`fixed`	`boolean`	Unwrap sites in clusters, that are flagged as fixed; options are `true` or `false`.
`connectivity`	`boolean`	Include connectivity in the resulting PDB using keyword CONECT; options are `true` or `false`.
crystal	boolean	Selects whether a read structure is a crystal; needed for small crystal structures with box crossing bonds; options are `true` and `false`.
element	option	Selects location of element (internally used for defining masses); options are `auto`, `element`, and `type`; `element` uses elements defined in PDB files; aromatic elements are all lower case (needed when reading united atom structures); `type` uses types as defined in PSF files, which is needed when reading coarse-grained structures; `auto` internally decides which of the previous options to use based on the choice of force field (see Section 5.11 [Field], page 106).

5.22.2 Usage

The `pdb` command is used to export data files in PDB and PSF format for convenient visualization with VMD (Visual Molecular Dynamics), a visualization conceived and supported by the Theoretical and Computational Biophysics Group at the University of Illinois at Urbana-Champaign. Textual PDB and PSF formats are ambiguous and are not self-defining, which creates the need for predefinition of the desired format. This port allows for commonly used force field modes. Cell dimensions are defined by addition of a `CRYST1` keyword to the PDB file.

5.22.3 Default

The default is given by

```
pdb             = {compress -> false, mode -> put, system -> 0,
                   forcefield -> auto, atomistic -> full, depth -> auto,
                   detect -> false, atom -> series, residue -> index,
                   segment -> index, vdw -> false, hexadecimal -> false,
                   cut -> false, pbc -> true, map -> false, unwrap -> true,
                   rigid -> true, fixed -> true, connectivity -> false,
                   crystal -> false, element -> auto};
```

5.22.4 References

1. VMD - Visual Molecular Dynamics, 'http://www.ks.uiuc.edu/Research/vmd/'
2. Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", *J. Molec. Graphics* **1996**, *14*, 33-38

5.23 Put

5.23.1 Syntax

```
put             = {name -> string, compress -> boolean, detail -> integer};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be exported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.
`detail`	`integer`	Sets level of detail in output file; a non-zero setting overrides the already set detail.

5.23.2 Usage

The `put` command is used to export data files generated by the current version of EMC, using a structured data format. Data files contain all necessary information to function as a restart file and are in a textual format.

5.23.3 Default

The default is given by

```
put             = {compress -> false, detail -> 0};
```

5.24 Rename

5.24.1 Syntax

```
rename          = {
  site          -> {src -> constant, dest -> constant},
  ...
  group         -> {src -> constant, dest -> constant},
  ...
  cluster       -> {src -> constant, dest -> constant},
  ...
  system        -> {src -> constant, dest -> constant},
  ...
};
```

Directive	Parameters	Description
site	struct	Defines multiple sites.
group	struct	Defines multiple groups.
cluster	struct	Defines multiple clusters.
system	struct	Defines multiple systems.

5.24.2 Usage

The `rename` command allows for renaming already existing site, group, cluster, or system ids.

5.24.3 Default

No default.

5.24.4 Examples

A few examples are given by

```
rename          = {
  site          -> {src -> bcc, dest -> a},
  site          -> {src -> fcc, dest -> b}
};

rename          = {
  site          -> {src -> fcc, dest -> a},
  cluster       -> {src -> fcc, dest -> surface}
};
```

5.25 Remove

5.25.1 Syntax

```
remove          = {n -> integer, mode -> option, nmin -> integer,
                   target -> index, region -> region};
```

Directive	Parameters	Description
n	integer	Selects number repeat units to be removed in each active system.
mode	option	Selects removal mode; viable options are `cluster`, `group`, and `site`.
target	index	Specifies the targeted group to remove; additionally a group site can be specified; a value of -1 one indicates `all`.
nmin	integer	Specifies the minimum allowable number of mobile repeat units that remain.
region	region	Specifies in which region in the simulation cell to remove (see Section 7.7 [Region], page 196).

5.25.2 Usage

The `remove` command deletes a maximum of `n` and a minimum of `nmin` groups specified by `group`, which fall within a specified `region`. All sites within the specified region are deleted, when no group is provided.

5.25.3 Default

```
remove          = {
  n             -> 0,
  mode          -> cluster,
  nmin          -> 0,
  target        -> {index -> -1, site -> -1},
  region        -> {shape -> spheroid, type -> relative, mode -> hard,
                    center -> {0,0,0}, h -> {0,0,0,0,0,0}}
};
```

5.25.4 Examples

Remove 100 water molecules as defined by group `water` in the whole simulation box,

```
remove          = {
  n             -> 100,
  target        -> {index -> water},
  region        -> {shape -> cuboid, radius -> {1, 1, 1}}
};
```

Note, that overspecification by the region radius of a factor of two ensures the inclusion of all particles in that box direction.

Remove 10 methyl groups as defined by group `methyl` in the center halve portion in the x direction of the simulation box,

```
remove          = {
  n             -> 10,
  target        -> {index -> methyl},
  region        -> {shape -> cuboid, radius -> {0.25, 1, 1}}
};
```

5.26 Reset

5.26.1 Syntax

```
reset           = {style -> option};
```

Directive	Parameters	Description
`style`	`integer`	Possible options are `simulation`, `statistics`, `profiles`, or `moves`.

5.26.2 Usage

The `reset` command allows for resetting entries and counters of either the whole simulation structure, or of separate subsections statistics, profiles, or moves.

5.26.3 Default

The default is given by

```
reset           = {style -> simulation};
```

5.27 Restart

5.27.1 Syntax

```
restart         = {name -> string, compress -> boolean, format -> string,
                    frequency -> integer, reset -> boolean};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be exported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.
`format`	`string`	Defines counter extention format.
`frequency`	`integer`	Selects output interval frequency in units of cycles.
`reset`	`boolean`	Resets simulation after writing restart file; options are `true` or `false`.

5.27.2 Usage

The `restart` command allows for writing of restart files at constant intervals during simulation, which safeguards long simulations in case of compute environment failure.

5.27.3 Default

The default is given by

```
restart         = {compress -> false, format -> "_%08ld",
                    frequency -> 10000, reset -> false};
```

5.28 Retype

5.28.1 Syntax

```
retype          = {
  mode          -> option,
  fraction      -> real,
  charge        -> boolean,
  n             -> {integer, ...},
  system        -> {constant, ...},
  source        -> {constant, ...},
  destination   -> {constant, ...},
  group         -> {index -> constant, site -> integer},
  inside        -> region,
  outside       -> region
};
```

Directive	Parameters	Description
`mode`	`option`	Sets the selection criteria for the fraction of selected sites; valid modes are: `random`.
`charge`	`boolean`	Selects charge transfer from mass paragraph to sites.
`fraction`	`real`	Defines the fraction of sites within the selection to be changed; values smaller than zero will be set to zero; values larger than one will be set to one; non-zero number of sites overrides fractions.
`n`	`integer`	Sets the number of sites to delete per defined targeted system; the first entry is taken for all systems when targeted systems are omited; fractions are taken when omited.
`system`	`constant`	Sets targeted system; can one or more systems; all systems are implied when omited.
`source`	`constant`	Sets source mass types; can be one or more types.
`destination`	`constant`	Sets destination mass types; can be one or more types; has to be either one element or the same number of elements as represented under the `source` directive.
`group`	`index`	Optional definition of group index and site for all destination types.
`inside`	`region`	Sets the inner boundary of the region in which mass types are to be changed (see Section 7.7 [Region], page 196); assumes domain center when ommited.
`outside`	`region`	Sets the outer boundary of the region in which mass types are to be changed (see Section 7.7 [Region], page 196); assumes whole domain when ommited.

5.28.2 Usage

The `retype` command allows for changing mass types from a source target consisting of either one or more types to a destination target consisting of one or more types. The destination target can either consist of one element or the same number of elements as the source target, in case the source target consists of more than one element. All groups, systems and regions are implied when none are supplied.

5.28.3 Default

The default is given by

```
retype          = {
  mode          -> random,
  fraction      -> 1
};
```

5.28.4 Examples

Note that the new site 'graft' needs to be defined in terms of mass and force field (see Section 5.32 [Sites], page 142, and Section 5.33 [Simulation], page 143). An example is given by

```
variables       = {r1 -> 1/2, r2 -> 1/2-0.75/(2^(2/3)*n)};
retype          = {
  source        -> fcc,
  destination   -> graft,
  inside        -> {shape -> spheroid,
                      center -> {0, 0, 0}, radius -> {r2, r2, r2}},
  outside       -> {shape -> spheroid,
                      center -> {0, 0, 0}, radius -> {r1, r1, r1}}
};
```

5.29 Run

5.29.1 Syntax

```
run             = {ncycles -> integer, nblocks -> integer,
                    cycle -> integer, seed -> integer};
```

Directive	Parameters	Description
`ncycles`	`integer`	Defines the number of cycles for which to run the simulation.
`nblocks`	`integer`	Defines the frequency with which to output intermediate energies in intervals of cycles.
`cycle`	`integer`	Defines the cycle counter at which to start the simulation; selects the cycle as defined by the input file when negative
`seed`	`integer`	Selects a random seed; the time is used when `seed` is greater than or equal to zero.

5.29.2 Usage

The `run` command is used to start a simulation. It allows for definition of simulation length and output frequency. Starting cycle and initial random seed can also be defined.

5.29.3 Default

The default is given by

```
run             = {cycle -> -1, seed -> 0};
```

5.30 Sample

5.30.1 Syntax

```
sample          = {
  cavity        -> struct,
  gr            -> struct,
  gyration      -> struct,
  ...
};
```

Directive	Parameters	Description
`cavity`	`struct`	Descriptor for cavity size sampling as defined by CESA (Cavity Energetic Sizing Algorithm)[1] (see Section 6.2 [Cavity], page 160).
`gr`	`struct`	Descriptor for radial distribution functions sampling (see Section 6.3 [Gr], page 165).
`gyration`	`struct`	Descriptor for radius of gyration sampling (see Section 6.4 [Gyration], page 170).

5.30.2 Usage

This variable style describes the control of sampling options (see Chapter 6 [Sampling Tools], page 156, for applications).

5.30.3 Default

By default, all sampling is deactivated.

5.30.4 References

1. P.J. in 't Veld, M.T. Stone, T.M. Trustkett, and I.C. Sanchez, "Liquid Structure via Cavity Size Distributions", *J. Phys. Chem. B* **2000**, *104*, 12028

5.31 Shell

5.31.1 Syntax

```
shell           = {command -> string};
```

Directive	Parameters	Description
`command`	`string`	Execute a shell command as specified by `string`.

5.31.2 Usage

The `shell` command is used to execute a shell command, which allows for file manuipulations during running. Any problems will be identified by a warning, thus not stopping the execution of the EMC script.

5.31.3 Default

The default is given by

```
shell           = {command -> ""};
```

5.32 Sites

5.32.1 Syntax

```
sites           = {
  site          -> {id -> constant,
                      reference -> integer, name -> string, mass -> real},
  ...
};
```

Directive	Parameters	Description
site	struct	Identifies site parameters; repeating structure.
id	constant	Sets site identity; can be either numerical or textual.
reference	integer	Sets a numerical reference; not used internally.
name	string	Sets a longer identifying name.
mass	real	Sets the mass in units set in units.

5.32.2 Usage

The sites command creates mass entries in the mass section of the types section of simulation. Multiple sites can be entered by using multiple instances of the site directive, separated by commas.

5.32.3 Default

The default is given by

```
sites           = {site -> {id -> 0, reference -> 0}};
```

5.32.4 Examples

A few examples are

```
sites           = {
  site          -> {id -> a, reference -> 0, name -> "colloid", mass -> 10}
};

sites           = {
  site          -> {id -> c,
                      reference -> 12, name -> "carbon", mass -> 12.011},
  site          -> {id -> h,
                      reference -> 1, name -> "hydrogen", mass -> 1.008},
  site          -> {id -> o,
                      reference -> 14, name -> "oxygen", mass -> 15.9994}
};
```

5.33 Simulation

5.33.1 Syntax

```
simulation      = {output -> {...}, units -> {...}, variables -> {...},
                   systems -> {...}, types -> {...}, moves -> {...},
                   profiles -> {...}};
```

Directive	Parameters	Description
`output`	`{...}`	Provides access to output parameters; possible directives include `detail`, `wide`, `expand`, `math`, `reduced`, `info`, `strict`, `warning`, `message`, and `debug`.
`units`	`{...}`	Provides access to internal units; possible directives include `mass`, `length`, `angstrom`, `angle`, `energy`, `kb`, `nav`, `charge`, `permittivity`, and `seed` (see Section 7.13 [Units], page 364).
`variables`	`{...}`	Provides access to changing predefined variables; used in conjunction with predefined variables in input files, allowing posteriori sizing of an input structure.
`types`	`{...}`	Provides access to all simulation-wide types; possible directives, which define force fields and their respective constants, include `boltzmann`, `charmm`, `coarse`, `coulomb`, `spline`, and `standard` (see Section 7.12 [Types], page 204).
`systems`	`{...}`	Provides access to all simulation-wide system settings; possible directives, which define e.g. system temperature or geometry, include `properties`; most system properties are derived output variables (e.g. `p`, `v`, `mass`, `nsites`, `nclusters`) rather than input variables (see Section 7.11 [Systems], page 202).
`moves`	`{...}`	Provides access to all moves; possible directives include `displace`, `endbridge`, `rebridge`, `reptate`, `rotate`, and `temper` (see Section 7.4 [Moves], page 182).
`profiles`	`{...}`	Provides acces to all profile definitions; possible directives include `density`, `force`, `mass`, and `order` (see Section 7.6 [Profiles], page 195).

5.33.2 Usage

The `simulation` command provides access to all variables and parameters defined within the simulation structure, of which the above table lists the main directives.

5.33.3 Examples

A few examples are (see data file for more suggestions)

```
simulation      = {output -> {detail -> 4}};
simulation      = {variables -> {lb -> 0.95*lb}};
```

```
simulation      = {types -> {standard -> {correct -> {active -> true}}}};
simulation      = {systems -> {
                    properties -> {
                      {id -> 0, t -> 298.15}, {id -> 1, t -> 315}}};
simulation      = {moves -> {displace -> {frequency -> 1}}};
simulation      = {profiles -> {density -> {active -> true}}};
```

5.34 Split

5.34.1 Syntax

```
split           = {system -> constant, direction -> option,
                   mode -> option, unwrap -> boolean, fraction -> real,
                   focus -> struct, region -> struct};
```

Directive	Parameters	Description
`system`	`constant`	Sets identity of system to which to add the build; can be either textual or numerical.
`direction`	`option`	Sets the direction in which the system is grown; options are `x`, `y`, and `z`.
`mode`	`option`	Sets the selection algorithm; options are `distance` and `random`.
`unwrap`	`boolean`	Sets wether to unwrap already existing clusters (only the first instance before a build should be set to `true`); options are `true` and `false`.
`fraction`	`real`	Defines the distribution fraction; valid values are between 0 and 1, inclusive the extremes.
`focus`	`struct`	Sets the selection to focus on (see see Section 7.3 [Focus], page 181).
`region`	`struct`	Sets the region to consider (see see Section 7.7 [Region], page 196).

5.34.2 Usage

The `split` command is used to redistribute clusters at the growing sides of a simulation box. Specific selection criteria can be set by defining a region in which the center of mass of targetted clusters need to reside. Additionally, a focussing selection can be specified based on site types, group ids, and cluster ids. Undefined `focus` and `region` directives imply considering all active sites in the system.

5.34.3 Default

The default is given by

```
split           = {system -> 0, direction -> x,
                   mode -> random, unwrap -> true, fraction -> 0.5};
```

5.35 Terminate

5.35.1 Syntax

```
terminate       = {mode -> option};
```

Directive	Parameters	Description
`mode`	`option`	Selects termination mode; currently only the option `all` is supported.

5.35.2 Usage

The `terminate` command is used to terminate free cluster ends created by `cut` and `remove` commands. Currently, terminators existing of only one site are supported. Future releases will include a group-based terminator.

5.35.3 Default

The default is given by

```
terminate       = {mode -> all};
```

5.36 Timing

5.36.1 Syntax

```
timing          = {style -> option};
```

Directive	Parameters	Description
`style`	`option`	Possible options are `none`, `show`, or `reset`.

5.36.2 Usage

The `timing` command allows for showing current timing, after which timing can be reset or allowed to accumulate further.

5.36.3 Default

The default is given by

```
timing          = {style -> show};
```

5.37 Traject

5.37.1 Syntax

```
traject         = {mode -> option, system -> integer,
                   name -> string, compress -> boolean, format -> string,
                   start -> integer, end -> integer, frequency -> integer,
                   append -> boolean, reset -> boolean, scale -> boolean,
                   unwrap -> boolean, pbc -> boolean};
```

Directive	Parameters	Description
mode	option	Sets the operation mode; viable options are get, put, and sample.
system	integer	Selects the system to export.
name	string	Defines the file name to be exported; an extention of .traject is assumed.
compress	boolean	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are true or false.
format	string	Defines counter extention format.
start	integer	Selects the frame at which to start reading; -1 indicates the last frame.
end	integer	Selects the frame at which to end reading; -1 indicates the last frame.
frequency	integer	Selects output interval frequency in units of cycles.
append	boolean	Appends each selected cycle to the end of the trajectory file; options are true or false.
reset	boolean	Resets output file; options are true or false.
scale	boolean	Scales the output using system box dimensions; options are true or false.
unwrap	boolean	Unwraps clusters in output; options are true or false.
pbc	boolean	Applies periodic boundary conditions; options are true or false.

5.37.2 Usage

The traject command allows for reading, writing, or analyzing of LAMMPS-style trajectory files. In read mode, a configuration from a trajectory can be loaded over already existing positions. In write mode, configurations are written at constant intervals during simulation. Output can be directed to either one or separate files by setting the append option to true or false respectively. Cycle counts are added to the file name when separate files are chosen, however they are omitted when the append option is set. In analyze mode, a full trajectory is analyzed using preset sampling settings (see Section 5.30 [Sample], page 140). An example can be found in ./examples/sample/cavity/traject.emc.

5.37.3 Default

The default is given by

```
traject         = {mode -> put,
                   compress -> false, format -> "_%08ld", system -> 0,
                   start -> 0, end -> -1, frequency -> 1000, append -> true,
                   reset -> true, scale -> false};
```

5.37.4 Examples

A few examples are given by

```
traject         = {name -> "test/polyethylene", compress -> true};
traject         = {name -> "water", format -> "_%ld", system -> 1,
                     frequency -> 10000, append -> false, scale -> false};
traject         = {name -> "cavity", frequency -> 1, mode -> analyze};
```

5.38 Translate

5.38.1 Syntax

```
translate       = {
  site          -> {constant, ...},
  group         -> {constant, ...},
  cluster       -> {constant, ...}
  delta         -> vector
};
```

Directive	Parameters	Description
sites	constant	Sets site or sites to focus on; refers to mass type constants.
groups	constant	Sets group or groups to focus on.
clusters		
delta	vector	Sets the displacement vector

5.38.2 Usage

The `translate` command facilitates translation of a selection of sites over a displacement vector `delta`. This scripting command can be helpful in translating surfaces constructed in the center of the simulation box to its edges.

5.38.3 Default

The default is given by

```
translate       = {
  vector        -> {0, 0, 0}
};
```

5.38.4 Examples

An example is given by

```
translate       = {
  cluster       -> surface,
  delta         -> {-lx/2, 0, 0}
};
```

which translates clusters called 'surface' to the lower edge of the simulation box. Note, that `lx` has to be provided by the user.

5.39 Types

5.39.1 Syntax

```
types           = {
  merge         -> boolean,
  virial        -> boolean,
  periodic      -> vector,
  neighbor      -> constant,
  stencil       -> constant,
  skin          -> real,
  shake         -> constant
  depth         -> integer,
  mass          -> struct,
  fields...
};
```

Directive	Parameters	Description
`merge`	`boolean`	Allows for merging force field constants upon input when true; options are `true` or `false`.
`virial`	`boolean`	Describes if virial calculations are included; options are `true` or `false`.
`periodic`	`vector`	Indicate periodicity with a three-element boolean vector with options `true` or `false`.
`neighbor`	`constant`	Describes what kind of neighbor list algorithm is used during pair interaction calculations; options are `sector` or `pair`.
`stencil`	`constant`	Describes the kind of stencil used during pair interaction calculations; options are `standard` or `multi`.
`skin`	`real`	Describes the skin used during pair interaction calculations; the skin is added to the pairwise cutoff.
`shake`	`constant`	Indicates the use of the SHAKE algorithm in subsequent codes (e.g. LAMMPS); valid options are `none`, `auto`, `hydrogen`, `water`, or `all`.
`depth`	`integer`	Maximum depth used for construction of ring structures during typing; allowed values are positive, where a value of 8 works in most ring cases; alternatively, an `auto` keyword allows for checking rings of unknown size; please note, that significant slow down occurs with the latter options for intricate ring systems.
`mass`	`struct`	Describes the site masses.
`fields`		Access to various field settings; see Section 7.12 [Types], page 204, for further information.

5.39.2 Usage

This variable style describes types (see Section 7.12 [Types], page 204, for further information) and allows direct access to all variables and parameters stored within the types structure.

5.39.3 Default

Unless otherwise stated, the default is given by

```
types           = {
  merge         -> false,
  virial        -> false,
  periodic      -> {true, true, true}
  neighbor      -> sector,
  stencil       -> standard,
  shake         -> none,
  depth         -> auto,
  skin          -> 0
};
```

By default, all force fields are deactivated.

5.40 Variables

5.40.1 Syntax

```
variables       = {variable -> string|integer|real, ...};
```

Directive	Parameters	Description
`variable`	`string`, `integer`, `real`	Sets or redefine a variable; currently supported format only include scalars of types `string`, `integer`, and `real`.

5.40.2 Usage

The `variable` command is used to define or override previously defined variables, which allows for e.g. resizing of simulation cells as defined in the original data file. Also, simulation-wide string variables can be set, which later can be used as file names, etc. Currently, only scalar variables are supported. Future releases will include structured variables as well.

5.40.3 Examples

Possible definitions include

```
variables       = {la -> 0.95*la, lb -> 1.05*lb};
variables       = {name -> "test/polyethylene"};
```

5.41 XYZ

5.41.1 Syntax

```
xyz             = {name -> string, compress -> boolean,
                   system -> integer, forcefield -> option,
                   cut -> boolean, detect -> boolean,
                   segment -> option, residue -> option,
                   pbc -> boolean, map -> boolean, unwrap -> boolean};
```

Directive	Parameters	Description
`name`	`string`	Defines the file name to be imported; an extention of .emc is assumed.
`compress`	`boolean`	Sets compression using Lempel-Ziv coding (LZ77); an extra extension of .gz is assumed; options are `true` or `false`.
`system`	`integer`	Selects system to be imported or exported
`forcefield`	`option`	Selects the desired force field; valid options are `none`, `standard`, or `charmm`.
`cut`	`boolean`	Selects cut mode: cut bonds that cross cell boundaries; options are `true` or `false`.
`detect`	`boolean`	Detect and assign atom and amino acid residue types; options are `true` or `false`.
`segment`	`option`	Sets options for segment representation; options are `detect` for detection, `index` for use of EMC cluster ids, or `series` for increments starting with the letter M.
`residue`	`option`	Sets options for residue representation; options are `detect` for detection of amino acid residues, `index` for use of EMC group ids, or `series` for increments starting with the letter R.
`pbc`	`boolean`	Applies periodic boundary conditions; options are `true` or `false`.
`map`	`boolean`	Selects mapping sites and box geometry according to a minimum image convention; options are `true` or `false`.
`unwrap`	`boolean`	Unwraps clusters to cross periodic boundaries with application of periodic boundary conditions to its center of mass when applicable; options are `true` or `false`.

5.41.2 Usage

The `xyz` command is used to export data files in XYZ format for convenient visualization with VMD (Visual Molecular Dynamics), a visualization conceived and supported by the Theoretical and Computational Biophysics Group at the University of Illinois at Urbana-Champaign. Textual XYZ formats are ambiguous and are not self-defining, which creates the need for predefinition of the desired format. This port allows for commonly used force field modes.

5.41.3 Default

The default is given by

```
xyz             = {compress -> false,
                   system -> 0, forcefield -> standard,
                   cut -> false, detect -> false,
                   segment -> index, residue -> index,
                   pbc -> true, map -> false, unwrap -> true};
```

5.41.4 References

1. VMD - Visual Molecular Dynamics, '`http://www.ks.uiuc.edu/Research/vmd/`'
2. Humphrey, W., Dalke, A. and Schulten, K., "VMD - Visual Molecular Dynamics", *J. Molec. Graphics* **1996**, *14*, 33-38

6 Sampling Tools

EMC offers the possibility of applying sampling either on the fly or by application to pregenerated trajectories (in `.traject` format). Available sampling algorithms include radii of gyration, pair correlation functions, cavity size distributions, energy and density profiles. The latter profiles offer a convolution option, which promotes smoothing out results by spreading point measurables over spherical volumes. Sampling is accessed through the scripting command `sample` (see Section 5.30 [Sample], page 140). Examples can be found in `./examples/build/sample`.

6.1 Bond

6.1.1 Syntax

```
gyration        = {
  id            -> integer,
  active        -> boolean,
  frequency     -> integer,
  source        -> struct,
  target        -> struct,
  binsize       -> real,
  distributions -> struct
};
```

Directive	Parameters	Description
`id`	`integer`	Method identifier for referencing subsequent alterations.
`active`	`boolean`	Indicates the state of the method; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the method, which is assumed to be a positive integer.
`source`, `target`	`struct`	Definition of clusters, groups, and sites as to which to focus analysis on (see Section 7.3 [Focus], page 181); both contributing types need to be included for a specific bond; all mobile and active sites are taken when not defined.
`binsize`	`real`	Sets the binsize of the resulting distributions.
`distributions`	`struct`	Distributions resulting from analysis; these distributions - only shown in .emc and exported .m files - cannot be influenced with scripting command `sample`.

6.1.2 Usage

The `bond` command is used to sample bond lengths, which are indicated by defining pairs withing `focus`. Resulting distributions are reported as frequency distributions, which can be exported by using the scripting command `export` (see Section 5.10 [Export], page 103). Averages are reported as part of the resulting distribution as shown in .emc and .m files. An example can be found in `./examples/build/sample/bond/` by running `setup.sh`.

6.1.3 Default

The default is given by

```
sample          = {
  id            -> 0,
  active        -> false,
  frequency     -> 1,
  source        -> {},
  target        -> {},
  binsize       -> 0.01
};
```

6.1.4 Example

The example as provided in `./examples/sample/bond/` creates a system of octane molecules using the TraPPE-UA force field (see Section 7.12.21 [TraPPE], page 356). The EMC build script is generated by `emc_setup.pl`, which uses `chemistry.csv` to define the octane chemistry (see Chapter 4 [Simulation Setup], page 8). The final system contains a total of 1000 beads. The `setup.sh` calls EMC to build and sample. The sampling script `sample.emc` is explained in the following. First, a set of variables are set, which are used later in the script.

```
variables       = {
  input         -> "example",
  output        -> "test",
  ncycles       -> 1000,
  nblocks       -> 100,
  frequency     -> 1
};
```

Next, the created structure `example.emc.gz` is loaded.

```
get             = {name -> input, compress -> true};
```

The structure should first be equilibrated before sampling commences. To this effect, a simulation is executed for `ncycles` cycles.

```
run             = {
  ncycles       -> ncycles,
  nblocks       -> nblocks
};
```

The bond lengths of bonds between CH3 and CH2 types and between CH2 types are to be sampled. To this end, sampling of bonds in defined in the `sample` paragraph. The TraPPE type for CH3 groups is `c4h3` and for CH2 groups `c4h2`.

```
sample          = {
  bond          -> {
    {
      id        -> 0,
      source    -> {sites -> c4h3},
      target    -> {sites -> c4h2},
      active    -> true, frequency -> frequency, binsize -> 0.001
    },
    {
      id        -> 1,
      source    -> {sites -> c4h2},
      target    -> {sites -> c4h2},
      active    -> true, frequency -> frequency, binsize -> 0.001
```

```
    }
  }
};
```

The simulations is continued for another `ncycles`. Sampling occurs during the execution of the simulation.

```
run             = {
  ncycles       -> ncycles,
  nblocks       -> nblocks
};
```

Trajectories can also be used for sampling analysis. To this effect, a trajectory can be loaded by using the `traject` keyword.

```
traject         = {name -> name, frequency -> 1, mode -> analyze};
```

Sampling of trajectories occurs when the `mode` option `analyze` is selected. The resulting distributions can be exported to either `math` and `csv` formats.

```
export          = {
  sample        -> {name -> "bond", type -> bond, style -> math},
  sample        -> {name -> "bond", type -> bond, style -> csv}
};
```

The equilibrated and resulting structure is stored in EMC format by means of

```
put             = {name -> output, compress -> true};
```

6.2 Cavity

6.2.1 Syntax

```
cavity          = {
  id            -> integer,
  active        -> boolean,
  frequency     -> integer,
  focus         -> struct,
  solver        -> option,
  record        -> string,
  separate      -> boolean,
  ninserts      -> integer,
  ntrials       -> integer,
  npoints       -> integer,
  tolerance     -> real,
  cutoff        -> real,
  binsize       -> real,
  distributions -> struct,
  nerrors       -> struct,
  zero          -> boolean,
  negative      -> boolean
};
```

Directive	Parameters	Description
`id`	`integer`	Method identifier for referencing subsequent alterations.
`active`	`boolean`	Indicates the state of the method; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the method, which is assumed to be a positive integer.
`focus`	`struct`	Defines sites, groups, or clusters to focus on (see Section 7.3 [Focus], page 181).
`solver`	`option`	Selects the solver used in finding local minima or saddle points in the energy surface; available options are `broyden` or `newton`.
`record`	`string`	Sets the output file name for the positions ans sizes of found cavities; ignored when no file name is set.
`separate`	`boolean`	Toggles whether to write recorded output in one single or separate files; options are `true` or `false`.
`ninserts`	`integer`	Sets the number of initially inserted points for finding local minima or saddle points.
`niterations`	`integer`	Sets the maximum number of iteration steps allowed in finding the local minima or saddle points.
`npoints`	`integer`	Sets the number of points used to determine the cavity size by means of the root of the local energy.

`tolerance`	`real`	Sets the tolerance for the zero force at the local minimum or saddle point.
`cutoff`	`real`	Sets the cutoff in Jacobian elements used to discriminate between minima and saddle points.
`binsize`	`real`	Sets the binsize of the resulting distributions.
`distributions`	`struct`	Distributions resulting from analysis; local minima and saddle points distributions are reported per system in alternating fashion.
`nerrors`	`struct`	Resulting bookkeeping of algorithm performance.
`zero`	`boolean`	Include negative sizes as zero size in distribution; negative sizes are ignored when `false`; superceedes `negative` when `true`; options are `true` or `false`.
`negative`	`boolean`	Include negative sizes in distribution; options are `true` or `false`.

6.2.2 Usage

The `cavity` command is used to sample cavity size distributions of all morphologies entailed by a simulation according to the algorithm as described by In 't Veld et al.[1] The algorithm uses either a Newton or Broyden iterative scheme in combination with a line search to determine local minima or saddle point in the surface of the energy landscape. This point is then used to grow a bead to such an extent, that the resulting energy equals zero. Distinction between minima and saddle points is made by requiring the Jacobian to be definate positive for minima. The definition of "definate positive" can be adjusted by means of the `cutoff` option. An example can be found in `./examples/build/sample/cavity/` by running `build.emc`. An example of trajectory analysis is given by `traject.emc` in the same examples directory.

6.2.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1,
  focus         -> {},
  solver        -> newton,
  record        -> "",
  separate      -> false,
  ninserts      -> 1000,
  ntrials       -> 200,
  npoints       -> 10,
  tolerance     -> 1e-8,
  cutoff        -> 1e-6,
  binsize       -> 0.01,
  zero          -> false,
  negative      -> false
};
```

6.2.4 Example

The following example samples the cavity size distribution of an FCC lattice, which consists of a tetrahedral and an octahedral cavity. Its scripting equivalent can be found in `./examples/sample/cavity/build.emc`. This example will be discussed in parts in the following. First, a number of variables are set,

```
variables       = {
  lattice       -> $root+"lib/fcc",
  output        -> "lattice",
  n             -> 4
};
```

Here the variable `lattice` describes the location of a predefined FCC lattice in EMC format. The final resulting positions are stored in filenames with starting with `lattice`. The variable `n` holds the number of replicas. An FCC lattice is obtained from the library directory by

```
get             = {name -> lattice};
```

after which it is replicated `n` times in all directions by

```
crystal         = {n -> {n, n, n}};
```

thus creating a superlattice. Subsequently, sampling of cavities using a Newton solver is turned on by

```
sample          = {
  cavity        -> {
    active      -> true,
    solver      -> newton,
    record      -> "cavity",
    binsize     -> 0.001,
    ninserts    -> 1000
  }
};
```

One thousand inserts are executed for each sampled structure. Cavity and saddle point positions are stored in file "cavity.pdb" by defining `record`. The resulting distribution has a bin size of 0.001. All omitted parameter definitions will internally be set to the above mentioned defaults. The selected FCC lattice does not include a force field definition, Force field parameters – which are not included in the selected FCC lattice – can be defined by using the `simulation` scripting command,

```
simulation      = {
  output        -> {debug -> false},
  units         -> {seed -> -1},
  types         -> {
```

```
    standard    -> {
      pair      -> {
        active  -> true,
        cutoff  -> 2.5,
        data    -> {
          {i0   -> 0, i1 -> 0, sigma -> 1, epsilon -> 1}
        }
      }
    }
  },
  moves         -> {
    displace    -> {active -> false}
  }
};
```

Here, the standard Lennard-Jones force field is chosen. Subsequently, displacement moves are switched off, thus not allowing particles to move. The seed value -1 triggers the use of the system clock as a seed value. A trajectory file is created through

```
traject         = {name -> "cavity", frequency -> 1};
```

The resulting configuration is run for 0 cycles in order to sample the cavity size distribution,

```
run             = {ncycles -> 0};
```

Once the cavity size distribution has been sampled, it is stored in both a Mathematica and a comma separated value format through

```
export          = {
  sample        -> {name -> "cavity", type -> cavity, style -> math},
  sample        -> {name -> "cavity", type -> cavity, style -> csv}
};
```

The superlattice is stored in both EMC native and PDB formats by means of

```
put             = {name -> output};
pdb             = {name -> output};
```

The above described script is run with

```
emc_linux build.emc 2>&1 | tee build.out
```

A VMD script called `cavity.vmd` has been provided in the same directory. This script allows for simulataneous visualization of particles, octahedral and tetrahedral cavities, demarcated with types `A` and `B` respectively. The script is called with

```
vmd -e cavity.vmd
```

The color of the solid particles identifies the size of the cavity. Note, that due to the statistical nature of the algorithm, not all cavities necessarily will be found. Thus, possibly not all cavities will be displayed.

6.2.5 References

1. P.J. in 't Veld, M.T. Stone, T.M. Trustkett, and I.C. Sanchez, "Liquid Structure via Cavity Size Distributions", *J. Phys. Chem. B* **2000**, *104*, 12028

6.3 Gr

6.3.1 Syntax

```
gr              = {
  id            -> integer,
  active        -> boolean,
  frequency     -> integer,
  source        -> struct,
  target        -> struct,
  cutoff        -> real,
  binsize       -> real,
  intra         -> struct,
  inter         -> struct,
  total         -> struct
};
```

Directive	Parameters	Description
`id`	`integer`	Method identifier for referencing subsequent alterations.
`active`	`boolean`	Indicates the state of the method; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the method, which is assumed to be a positive integer.
`source`, `target`	`struct`	Definition of source and target clusters, groups, and sites (see Section 7.3 [Focus], page 181); all mobile and active sites are taken when not defined.
`cutoff`	`real`	Sets the extent to which distances are evaluated; a zero value will result in initialization with the maximum cutoff resulting from active pairwise potential definitions.
`binsize`	`real`	Sets the binsize of the resulting distributions.
`intra`, `inter`, `total`	`struct`	Distributions resulting from analysis, reporting intramolecular, intermolecular, and all distances resulting from pairwise combinations as specified by source and target structures.

6.3.2 Usage

The `gr` command is used to sample radial distribution functions of all morphologies entailed by a simulation. Intramolecular, intermolecular, and total distributions are sampled based on the specified source and target definitions. Resulting distributions are reported as frequency distributions, which can be exported by using the scripting command `export` (see Section 5.10 [Export], page 103). An example can be found in `./examples/build/sample/gr/` by running `setup.sh`.

6.3.3 Default

The default is given by

```
sample          = {
```

```
  id            -> 0,
  active        -> false,
  frequency     -> 1,
  source        -> {},
  target        -> {},
  cutoff        -> 0,
  binsize       -> 0.01
};
```

6.3.4 Pair Correlation Functions

The following has been borrowed from Wikipedia[1]. In statistical mechanics, the radial distribution function, (or pair correlation function) $g(r)$ in a system of particles (atoms, molecules, colloids, etc.), describes how density varies as a function of distance from a reference particle.

If a given particle is taken to be at the origin O, and if $\rho = N/V$ is the average number density of particles, then the local time-averaged density at a distance r from O is $\rho g(r)$. This simplified definition holds for a homogeneous and isotropic system. A more general case will be considered below.

In simplest terms it is a measure of the probability of finding a particle at a distance of r away from a given reference particle, relative to that for an ideal gas. The general algorithm involves determining how many particles are within a distance of r and $r + dr$ away from a particle. This general theme is depicted to the right, where the red particle is our reference particle, and blue particles are those which are within the circular shell, dotted in orange.

The RDF is usually determined by calculating the distance between all particle pairs and binning them into a histogram. The histogram is then normalized with respect to an ideal gas, where particle histograms are completely uncorrelated. For three dimensions, this normalization is the number density of the system multiplied by the volume of the spherical shell, which mathematically can be expressed as $g(r)_I = 4\pi r^2 \rho dr$, where ρ is the number density.

Given a potential energy function, the radial distribution function can be computed either via computer simulation methods like the Monte Carlo method, or via the Ornstein-Zernike equation, using approximative closure relations like the Percus-Yevick approximation or the Hypernetted Chain Theory. It can also be determined experimentally, by radiation scattering techniques or by direct visualization for large enough (micrometer-sized) particles via traditional or confocal microscopy.

The radial distribution function is of fundamental importance since it can be used, using the Kirkwood–Buff solution theory, to link the microscopic details to macroscopic properties. Moreover, by the reversion of the Kirkwood-Buff theory, it is possible to attain the microscopic details of the radial distribution function from the macroscopic properties.

6.3.5 Definition

Consider a system of N particles in a volume V (for an average number density $\rho = N/V$) and at a temperature T (let us also define $\beta = \frac{1}{kT}$). The particle coordinates are $\vec{r}_i$, with $i = 1, ..., N$. The potential energy due to the interaction between particles is $U_N(\vec{r}_1, ..., \vec{r}_N)$ and we do not consider the case of an externally applied field.

The appropriate averages are taken in the canonical ensemble (N, V, T), with $Z_N = \int ... \int e^{-\beta U_N} d\vec{r}_1 \; ... \; d\vec{r}_N$ the configurational integral, taken over all possible combinations of particle positions. The probability of an elementary configuration, namely finding particle 1 in $d\vec{r}_1$, particle 2 in $d\vec{r}_2$, etc. is given by

$$P^{(N)}\left(\vec{r}_1, \; ..., \vec{r}_N\right) \; d\vec{r}_1 \; ... \; d\vec{r}_N = \frac{e^{-\beta U_N}}{Z_N} \; d\vec{r}_1 \; ... \; d\vec{r}_N \; .$$

The total number of particles is huge, so that $P^{(N)}$ in itself is not very useful. However, one can also obtain the probability of a reduced configuration, where the positions of only $n < N$ particles are fixed, in $\vec{r}_1 \; ..., \; \vec{r}_n$, with no constraints on the remaining $N-n$ particles. To this end, one has to integrate the above equation over the remaining coordinates $\vec{r}_{n+1}, \; ..., \; \vec{r}_N$:

$$P^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n) = \frac{1}{Z_N} \int ... \int e^{-\beta U_N} \; d\vec{r}_{n+1} \; ... \; d\vec{r}_N.$$

The particles being identical, it is more relevant to consider the probability that any n of them occupy positions $\vec{r}_1, ..., \; \vec{r}_n$ in any permutation, thus defining the n-particle density

$$\rho^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n) = \frac{N!}{(N-n)!} P^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n).$$

For $n = 1$, this equation gives the one-particle density which, for a crystal, is a periodic function with sharp maxima at the lattice sites. For a (homogeneous) liquid, it is independent of the position $\vec{r}_1$ and equal to the overall density of the system:

$$\frac{1}{V} \int \rho^{(1)}(\vec{r}_1) \; d\vec{r}_1 = \rho^{(1)} = \frac{N}{V} = \rho.$$

It is now time to introduce a correlation function $g^{(n)}$ by

$$\rho^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n) = \rho^n g^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n).$$

$g^{(n)}$ is called a correlation function, since if the atoms are independent from each other $\rho^{(n)}$ would simply equal ρ^n and therefore $g^{(n)}$ corrects for the correlation between atoms. From the latter two equations it follows that

$$g^{(n)}(\vec{r}_1, \; ..., \; \vec{r}_n) = \frac{V^n N!}{N^n (N-n)!} \cdot \frac{1}{Z_N} \; \int ... \int e^{-\beta U_N} \; d\vec{r}_{n+1} \; ... \; d\vec{r}_N.$$

6.3.6 Relations Involving g(r)

6.3.6.1 Structure Factor

The second-order correlation function $g^{(2)}(\vec{r}_1, \vec{r}_2)$ is of special importance, as it is directly related (via a Fourier transform) to the structure factor of the system and can thus be determined experimentally using X-ray diffraction or neutron diffraction. If the system consists of spherically symmetric particles, $g^{(2)}(\vec{r}_1, \vec{r}_2)$ depends only on the relative distance between them, $\vec{r}_{12} = \vec{r}_2 - \vec{r}_1$. We will drop the sub- and superscript: $g(\vec{r}) \equiv g^{(2)}(\vec{r}_{12})$. Taking particle 0 as fixed at the origin of the coordinates, $\rho g(\vec{r}) \; dr = dn(\vec{r})$ is the number

of particles (among the remaining $N-1$) to be found in the volume $d\vec{r}$ around the position $\vec{r}$. We can formally count these particles as $dn(\vec{r}) = \langle \sum_{i\neq 0} \delta(\vec{r}-\vec{r}_i)\rangle d\vec{r}$, with $\langle ...\rangle$ the ensemble average, yielding

$$g(\vec{r}) = \tfrac{1}{\rho}\langle \textstyle\sum_{i\neq 0} \delta(\vec{r}-\vec{r}_i)\rangle = V\tfrac{N-1}{N}\,\langle \delta(\vec{r}-\vec{r}_1)\rangle,$$

where the second equality requires the equivalence of particles 1, ..., $N-1$. The formula above is useful for relating $g(\vec{r})$ to the static structure factor $S(\vec{q})$, defined by $S(\vec{q}) = 1/N\langle \sum_{ij} e^{-i\vec{q}(\vec{r}_i-\vec{r}_j)}\rangle$, since we have

$$\begin{aligned} S(\vec{q}) &= 1 + \tfrac{1}{N}\langle \textstyle\sum_{i\neq j} e^{-i\vec{q}(\vec{r}_i-\vec{r}_j)}\rangle = \\ &= 1 + \tfrac{1}{N}\left\langle \textstyle\int_V e^{-i\vec{q}\vec{r}} \sum_{i\neq j} \delta\left[\vec{r}-(\vec{r}_i-\vec{r}_j)\right] d\vec{r}\right\rangle = \\ &= 1 + \tfrac{N(N-1)}{N}\textstyle\int_V e^{-i\vec{q}\vec{r}}\,\langle \delta(\vec{r}-\vec{r}_1)\rangle\, d\vec{r}\,, \end{aligned}$$

and thus

$$S(\vec{q}) = 1 + \rho \textstyle\int_V e^{-i\vec{q}\vec{r}} g(\vec{r})\, d\vec{r}\,,$$

proving the Fourier relation alluded to above. This equation is only valid in the sense of distributions, since $g(\vec{r})$ is not normalized: $\lim_{r\to\infty} g(\vec{r}) = 1$, so that $\int_V d\vec{r} g(\vec{r})$ diverges as the volume V, leading to a Dirac peak at the origin for the structure factor. Since this contribution is inaccessible experimentally we can subtract it from the equation above and redefine the structure factor as a regular function,

$$S'(\vec{q}) = S(\vec{q}) - \rho\delta(\vec{q}) = 1 + \rho \textstyle\int_V e^{-i\vec{q}\vec{r}}[g(\vec{r})-1]\, d\vec{r}\,.$$

Finally, we rename $S(\vec{q}) \equiv S'(\vec{q})$ and, if the system is a liquid, we can invoke its isotropy,

$$S(q) = 1 + \rho \textstyle\int_V e^{-i\vec{q}\vec{r}}[g(r)-1]\, d\vec{r} = 1 + 4\pi\rho\frac{1}{q}\int r\sin(qr)[g(r)-1]\, dr\,.$$

6.3.6.2 Compressibility Equation

Evaluating the latter equation for $q = 0$ while using the relation between the isothermal compressibility χ_T and the structure factor at the origin yields the compressibility equation:

$$\rho kT\chi_T = kT\left(\tfrac{\partial\rho}{\partial p}\right) = 1 + \rho\textstyle\int_V [g(r)-1]\, d\vec{r}.$$

6.3.6.3 Potential of Mean Force

It can be shown that the radial distribution function is related to the two-particle potential of mean force $w^{(2)}(r)$ by[2]

$$g(r) = \exp\left[-\tfrac{w^{(2)}(r)}{kT}\right].$$

6.3.6.4 Energy Equation

If the particles interact via identical pairwise potentials[3],

$$U_N = \sum_{i>j=1}^{N} u(|\vec{r}_i - \vec{r}_j|),$$

the average internal energy per particle is

$$\frac{\langle E \rangle}{N} = \frac{3}{2}kT + \frac{\langle U_N \rangle}{N} = \frac{3}{2}kT + \frac{1}{2}\rho \int_V d\vec{r}\ u(r)g(r,\rho,T).$$

6.3.6.5 Pressure Equation of State

Developing the virial equation yields the pressure equation of state,

$$p = \rho kT - \frac{1}{6}\rho^2 \int_V\ r\ g(r,\ \rho,\ T)\frac{du(r)}{dr}d\vec{r}.$$

6.3.6.6 Thermodynamic Properties

The radial distribution function is an important measure because several key thermodynamic properties, such as potential energy and pressure can be calculated from it. For a 3-D system where particles interact via pairwise potentials, the potential energy of the system can be calculated as follows[4]

$$PE = 2\pi\rho N \int_0^\infty r^2 u(r)g(r)dr,$$

where N is the number of particles in the system, ρ is the number density, $u(r)$ is the pair potential. The pressure of the system can also be calculated by relating the 2nd virial coefficient to $g(r)$. The pressure can be calculated as follows

$$P = \rho k_B T - \frac{2}{3}\pi\rho^2 \int_0^\infty r^3 g(r)\frac{du(r)}{dr}dr,$$

where T is the temperature and k_B is Boltzmann's constant. Note that the results of potential and pressure will not be as accurate as directly calculating these properties because of the averaging involved with the calculation of $g(r)$.

6.3.7 References

1. N. N., Radial Distribution Function, Wikipedia (1/1/2016).
2. Chandler, D., Introduction to Modern Statistical Mechanics. Chapter 7.3, Oxford University Press (1987).
3. Hansen, J. P. and McDonald, I. R., Theory of Simple Liquids (3rd ed.), Academic Press (2005).
4. Frenkel, D. and Smit, B., Understanding Molecular Simulation from Algorithms to Applications (2nd ed.), Academic Press (2002).

6.4 Gyration

6.4.1 Syntax

```
gyration        = {
  id            -> integer,
  active        -> boolean,
  frequency     -> integer,
  focus         -> struct,
  cutoff        -> real,
  binsize       -> real,
  distributions -> struct
};
```

Directive	Parameters	Description
`id`	`integer`	Method identifier for referencing subsequent alterations.
`active`	`boolean`	Indicates the state of the method; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the method, which is assumed to be a positive integer.
`focus`	`struct`	Definition of clusters, groups, and sites as to which to focus analysis on (see Section 7.3 [Focus], page 181); all mobile and active sites are taken when not defined.
`cutoff`	`real`	Sets the extent to which distances are evaluated; a zero value will result in initialization with the maximum cutoff resulting from active pairwise potential definitions.
`binsize`	`real`	Sets the binsize of the resulting distributions.
`distributions`	`struct`	Distributions resulting from analysis; these distributions - only shown in .emc and exported .m files - cannot be influenced with scripting command `sample`.

6.4.2 Usage

The `gyration` command is used to sample radius of gyration distributions of all morphologies entailed by a simulation. Resulting distributions are reported as frequency distributions, which can be exported by using the scripting command `export` (see Section 5.10 [Export], page 103). Averages are reported as part of the resulting distribution as shown in .emc and .m files. An example can be found in `./examples/build/sample/gyration/` by running `setup.sh`.

6.4.3 Default

The default is given by

```
sample          = {
  id            -> 0,
  active        -> false,
  frequency     -> 1,
  focus         -> {},
```

```
  cutoff        -> 0,
  binsize       -> 0.01
};
```

6.4.4 Theory

The following has been borrowed from Wikipedia[1]. In polymer physics, the radius of gyration is used to describe the dimensions of a polymer chain. The radius of gyration of a particular molecule at a given time is defined as:

$$R_g^2 = \frac{1}{N^2} \sum_{i=1,j>i}^{N} (\vec{r}_i - \vec{r}_j)^2,$$

where N represents the number of particles in the molecule and vectors $\vec{r}_i$ and $\vec{r}_j$ denote particle positions. Since the chain conformations of a polymer sample are quasi infinite in number and constantly change over time, the "radius of gyration" discussed in polymer physics must usually be understood as a mean over all polymer molecules of the sample and over time. That is, the radius of gyration which is measured as an average over time or ensemble:

$$\langle R_g^2 \rangle = \frac{1}{N^2} \langle \sum_{i=1,j>i}^{N} (\vec{r}_i - \vec{r}_j)^2 \rangle,$$

where the angular brackets $\langle ... \rangle$ denote the ensemble average. An entropically governed polymer chain (i.e. in so called theta conditions) follows a random walk in three dimensions. The radius of gyration for this case is given by

$$R_g = \frac{1}{\sqrt{6}} \sqrt{N}\ a$$

Note that although aN represents the contour length of the polymer, a is strongly dependent of polymer stiffness and can vary over orders of magnitude. N is reduced accordingly. One reason that the radius of gyration is an interesting property is that it can be determined experimentally with static light scattering as well as with small angle neutron- and x-ray scattering. This allows theoretical polymer physicists to check their models against reality. The hydrodynamic radius is numerically similar, and can be measured with Dynamic Light Scattering (DLS).

6.4.5 References

1. N. N., Radius of Gyration, Wikipedia (1/1/2016).

6.5 Interaction

6.5.1 Syntax

```
interaction     = {
  id            -> integer,
  active        -> boolean,
  frequency     -> integer,
  cutoff        -> real,
  mode          -> option,
  coulomb       -> option,
  bias          -> option,
  weight        -> option,
  ntrials       -> integer,
  nvolumes      -> integer,
  ninits        -> integer,
  nwidth        -> integer,
  source        -> struct,
  target        -> struct,
  mass          -> real,
  dvolume       -> real,
  volume1       -> struct,
  volume2       -> struct,
  denergy       -> real,
  energy        -> struct
};
```

Directive	Parameters	Description
`id`	`integer`	Method identifier for referencing subsequent alterations.
`active`	`boolean`	Indicates the state of the method; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the method, which is assumed to be a positive integer.
`mode`	`global`	Sets the desired pairwise potential cut off mode; options are `global`, `individual`, and `repulsive`
`coulomb`	`none`	Sets the desired type of coulombic interactions; options are `none`, `cut`, `switch`, and `long`.
`bias`	`none`	Sets the bias used during interaction sampling; options are `none`, `energy`, and `frequency`.
`weight`	`none`	Sets the weighting used for populating the resulting interaction distribution; options are `none`, `boltzmann`, and `montecarlo`.
`ntrials`	`0`	Sets the number of energetic interaction sampling trials.
`nvolumes`	`100`	Sets the number of volume sampling trials.

`ninits`	`1000`	Sets the number of trials for populating the initial distribution as used by `energy` and `frequency` biased sampling.
`nwidth`	`1`	Sets the spread used during sampling for accessing non-initialized bins resulting from the previous option.
`source`, `target`	`struct`	Definition of source and target clusters, groups, and sites (see Section 7.3 [Focus], page 181) setting the two interaction species; all mobile and active sites are taken when not defined.
`cutoff`	`real`	Sets the extent to which distances are evaluated; a zero value will result in initialization with the maximum cutoff resulting from active pairwise potential definitions.
`dvolume`	`real`	Sets the binsize of the resulting distributions.
`volume1`, `volume2`	`real`	Resulting distribution of source and target volume analysis; produces the volume distribution of the two interacting species.
`denergy`	`real`	Sets the binsize of the resulting energy distributions.
`energy`	`struct`	Distributions resulting from analysis, reporting pairwise interaction as specified by source and target structures.

6.5.2 Usage

The `interaction` command is used to sample the pairwise potential of mean force, which can either be calculated from provided or from internally generated structures. Internally generated structures are sampled over distances within the specified `cutoff` and over randomly chosen orientations. Resulting pairwise interactions can be weighted by several weighting techniques. Resulting samples per bin can be controlled by different biasing schemes. Volumetric and energetic distributions are sampled based on the specified source and target definitions. Resulting distributions are reported as frequency distributions, which can be exported by using the scripting command `export` (see Section 5.10 [Export], page 103).

6.5.3 Default

The default is given by

```
sample          = {
  id            -> 0,
  active        -> false,
  frequency     -> 1,
  mode          -> option,
  coulomb       -> option,
  bias          -> none,
  weight        -> none,
  ntrials       -> 0,
  nvolumes      -> 100,
  ninits        -> 1000,
  nwidth        -> 1,
  source        -> {},
```

```
  target        -> {},
  mass          -> 0,
  cutoff        -> 0,
  dvolume       -> 0.01,
  denergy       -> 0.01
};
```

6.6 Profiles

Density, pressure, and energy profile explanation.

6.7 Examples

Choice examples.

7 Variable Descriptions

EMC uses hierarchical variables, which can be recursive in character. This chapter describes the syntax of variables that are used often within the context of the EMC scripting language. Currently, variables use a representation similar to Mathematica.

Mathematical operations supported for scalar variable types `integer` and `real` are

Operator	Application	Description
`+`	`a + b`	Addition
`-`	`a - b`	Subtraction
`*`	`a * b`	Multiplication
`/`	`a / b`	Division
`^`	`a ^ b`	Power
`?`	`q ? a : b`	Logical; select `a` when `q` is true, `b` otherwise

Mathematical functions supported for scalar variable types `integer` and `real` are

Operator	Application	Description
`acos`	`acos(x)`	Arccosine using a radial basis
`asin`	`asin(x)`	Arcsine using a radial basis
`atan`	`atan(x)`	Arctangent using a radial basis
`cos`	`cos(x)`	Cosine using a radial basis
`eval`	`eval(expr)`	Evaluate a string expression into a value
`exp`	`exp(x)`	Exponential
`int`	`int(x)`	Floor of a real number
`log`	`log(x)`	Natural log
`sin`	`sin(x)`	Sine using a radial basis
`sqrt`	`sqrt(x)`	Square root
`tan`	`tan(x)`	Tangent using a radial basis
`mass`	`mass(index)`	Mass of a predefined group index
`mtotal`	`mtotal(focus)`	Mass of sites in selection as defined by focus (See Section 7.3 [Focus], page 181)
`nclusters`	`nclusters( focus)`	Number of clusters in selection as defined by focus (See Section 7.3 [Focus], page 181)
`nsites`	`nsites(index)`	Number of sites of a predefined group index
`ntotal`	`ntotal(focus)`	Number of sites in selection as defined by focus (See Section 7.3 [Focus], page 181)
`type`	`type(index)`	Determine type number of site index (used for transfer to e.g. LAMMPS input scripts)
`vsites`	`vsites(focus)`	Volume captured by selection as defined by focus (See Section 7.3 [Focus], page 181)
`vtotal`	`vtotal(focus)`	Total system volume as defined by system selection (See Section 7.3 [Focus], page 181)

Note, that groups must have been defined (see Section 5.17 [Groups], page 117) and a force field must have been applied (see Section 5.11 [Field], page 106) in order for mass(index) and nsites(index) to return correct values. See also `scripts/emc_setup.pl`.

Reverved variable names are represented by the following constants

Constant	**Type**	**Description**
`$root`	`string`	EMC root directory
`$arg#`	varies	command line argument
`pi`	`real`	number: 3.14159265358979323846264338327950288
`e`	`real`	number: 2.71828182845904523536028747135266250
`true`	`integer`	number: 0
`false`	`integer`	number: 1
`null`	`integer`	number: -1

Internally defined constants are booleans `true` and `false`, mathematical constants `pi` and `e`, internal variable `null`, and the program's root directory `$root`. For the latter, it is assumed, that the `emc` executable resides in `$root/bin`. All other locations will result in erroneous behavior of `$root`. Command line arguments not preceeded by a `"-"` are accessed through the prefix `$arg` followed by a number, starting at 0. For example

```
variables = {name -> $arg0};
```

sets the variable `name` to the first command line argument. In case of the command

```
emc_${HOST} build.emc atoms
```

the variable `name` in the above example would be set to the text `"atoms"`. Note, that – in this example – calling `$arg1` and subsequent non-exisiting arguments as a variable will result in an error due to the fact, that `$arg1` is not defined on the command line and therefore also not internally. Addition operations using `a + b`, where either `a` or `b` is of variable type `string`, will result in a string. For example

```
"text" + 1 + 1 := "text11"
```

but

```
"text" + (1 + 1) := "text2"
```

Previously defined variables can also be included

```
a = 2^2
"text" + a := "text4"
```

or

```
a = 2*3
dir = "/home/user/text"
dir + "_" + a := "/home/user/text_6"
```

7.1 Constants

7.1.1 Syntax

```
constants       = {
  systems       -> {constant, ...},
  clusters      -> {constant, ...},
  groups        -> {constant, ...},
  sites         -> {constant, ...}
};
```

Directive	Parameters	Description
systems	constant	Adds a constant to the systems category; constants are expected to be alpha-numerical.
clusters	constant	Adds a constant to the clusters category; constants are expected to be alpha-numerical.
groups	constant	Adds a constant to the groups category; constants are expected to be alpha-numerical.
sites	constant	Adds a constant to the sites category; constants are expected to be alpha-numerical.

7.1.2 Usage

This variable style describes constants. The style is additive when used in combinations with see Section 5.40 [Variables], page 153.

7.1.3 Default

Unless otherwise stated, the default is given by

```
constants       = {
  systems       -> {},
  clusters      -> {},
  groups        -> {},
  sites         -> {}
};
```

7.2 System Flags

7.2.1 Syntax

```
flag            -> option
```

Valid options are, when set,

Directive	Parameters
`ignore`	Ignore any missing contribution and do not generate any output
`complete`	Complete any missing contribution
`warn`	Generate a warning when a contribution is missing
`empty`	Create an empty entry when a contribution is missing
`error`	Exit on errors resulting from missing contributions; all missing contributions will be listed before an exit on error occurs

7.2.2 Default

Unless otherwise stated, the default is given by

```
flag            -> error
```

7.3 Focus

7.3.1 Syntax

```
focus           = {
  clusters      -> {constant, ...},
  groups        -> {constant, ...},
  sites         -> {constant, ...},
  systems       -> {constant, ...},
  ntrials       -> integer
};
```

Directive	Parameters	Description
`clusters`	`constant`	Refers to cluster constants as defined by the `constants` table.
`groups`	`constant`	Refers to group constants as defined by the `constants` table.
`ntrials`	`integer`	Defines the number of trials used to determine volume; volume is determined by means of Monte Carlo integration.
`sites`	`constant`	Refers to site constants as defined by the `constants` table.
`systems`	`constant`	Refers to system constants as defined by the `constants` table.

7.3.2 Usage

Defines a system sites subset by means of their constants identifiers (see Section 7.1 [Constants], page 179).

7.3.3 Default

The default is described by an empty list for all three contributors,

```
focus           = {
  clusters      -> {},
  groups        -> {},
  sites         -> {},
  systems       -> {},
  ntrials       -> 10000
};
```

7.4 Moves

7.4.1 Syntax

```
moves           = {
  ncycles       -> integer,
  cycle         -> integer,
  move          -> integer,
  body          -> struct,
  deform        -> struct,
  displace      -> struct,
  endbridge     -> struct,
  migrate       -> struct,
  rebridge      -> struct,
  reptate       -> struct,
  rotate        -> struct,
  surface       -> struct,
  temper        -> struct
};
```

Directive	Parameters	Description
ncycles	integer	Sets the total number of simulation cycles.
cycle	integer	Sets the current simulation cycle
move	integer	Sets the current simulation move.
body	struct	Settings for displacing sites inside a body.
deform	struct	Settings for system box deformation (see Section 7.4.6 [Displace], page 185).
displace	struct	Settings for displacing sites in a system box.
endbridge	struct	Settings for recombining cluster ends in a system box.
migrate	struct	Settings for migrating short branches.
rebridge	struct	Settings for rebridging clusters in a system box.
reptate	struct	Settings for reptating cluster ends in a system box.
rotate	struct	Settings for rotating cluster ends in a system box.
surface	struct	Settings for displacing sites on a body surface.
temper	struct	Settings for parallel tempering between systems.

7.4.2 Usage

This variable style describes moves. These moves encompass standard and advanced Monte Carlo moves.[1,2]

7.4.3 Default

Unless otherwise stated, the default is given by

```
moves           = {
  ncycles       -> 0,
  cycle         -> 0,
  move          -> 0
```

```
};
```

By default, all moves are activated with zero frequency, with the exception of the displacement move (see Section 7.4.6 [Displace], page 185), which has a frequency of one.

7.4.4 References

1. P.J. in 't Veld, M. Hütter, and G.C. Rutledge, "Temperature-Dependent Thermal and Elastic Properties of the Interlamellar Phase of Semicrystalline Polyethylene by Molecular Simulation", *Macromolecules* **2006**, *39*, 439
2. V. Kumar, C.R. Locker, P.J. in 't Veld, G.C. Rutledge, "Effect of Short Chain Branching on the Interlamellar Structure of Semicrystalline Polyethylene", *Macromolecules* **2017**, *50*, 1206

7.4.5 Deform Move

7.4.5.1 Syntax

```
deform  = {
  active        -> boolean,
  mode          -> option,
  frequency     -> integer,
  n             -> integer,
  dmax          -> real,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`mode`	`option`	Set the mode of deformation; valid options are `isotropic`, `shape`, `full`, `xx`, `yx`, `yy`, `zx`, `zy`, and `zz`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`...`		
`n`	`integer`	Number of masses available in the simulation; controlled internally: output only.
`accept`	`struct`	Array of accumulative accepted and total trials; output only.

7.4.5.2 Usage

The `deform` move alters the box geometry based the `mode` and the selected pressure as set in `systems` (see Section 7.11 [Systems], page 202).

7.4.5.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1
};
```

7.4.6 Displace

7.4.6.1 Syntax

```
displace        = {
  active        -> boolean,
  frequency     -> integer,
  dlimit        -> real,
  nsites        -> integer,
  couple        -> boolean,
  n             -> integer,
  dmax          -> real,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`dlimit`	`real`	Set the maximum displacement limit.
`couple`	`boolean`	Couple all individual acceptances when internally determining displacement limits.
`nsites`	`integer`	Set the number of sites to be displaced within one move.
`n`	`integer`	Number of masses available in the simulation; controlled internally.
`dmax`	`real`	Array of adapted displacement limits based on simulation progress; internally determined using a feedback loop; output only.
`accept`	`struct`	Array of accumulative accepted and total trials.

7.4.6.2 Usage

The `displace` move is used to displace sites by means of perturbation. The target acceptance is set to 50%.

7.4.6.3 Default

The default is given by

```
sample          = {
  active        -> true,
  couple        -> false,
  frequency     -> 1,
  nsites        -> 1
};
```

7.4.7 Endbridge

7.4.7.1 Syntax

```
endbridge       = {
  active        -> boolean,
  frequency     -> integer,
  target        -> constant,
  nmin          -> integer,
  nmax          -> integer,
  tolerance     -> real,
  dmin          -> real,
  dmax          -> real,
  n             -> integer,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`target`	`constant`	Target group.
`nmin`	`integer`	Minimum chain length after end-bridging; valid values are values larger or equal than three.
`nmax`	`integer`	Maximum chain length after end-bridging; chain length is unlimited when zero.
`tolerance`	`real`	Tolerance; currently not in use.
`dmin`	`real`	Minimum distance between parent chain end and child end-bridging candidate chain.
`dmax`	`real`	Maximum distance between parent chain end and child end-bridging candidate chain.
`n`	`integer`	Number of masses available in the simulation; controlled internally: output only.
`accept`	`struct`	Array of accumulative accepted and total trials; output only.

7.4.7.2 Usage

The `endbridge` move allows for recombination of chain ends with other chains; currently, only linear non-branched chains are allowed.

7.4.7.3 Default

The default is given by

```
sample          = {
  active        -> false,
```

```
  frequency     -> 1,
  nmin          -> 3
};
```

7.4.8 Migrate

7.4.8.1 Syntax

```
migrate         = {
  active        -> boolean,
  frequency     -> integer,
  target        -> constant,
  radius        -> real,
  n             -> integer,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`target`	`constant`	Target type for moving the branch to.
`radius`	`real`	Select sites with target types within radius from type on backbone; selects all available sites in the system when the radius equals zero.
`n`	`integer`	Number of masses available in the simulation; controlled internally: output only.
`accept`	`struct`	Array of accumulative accepted and total trials; output only.

7.4.8.2 Usage

The `migrate` move is used to migrate branches on the backbone of a chain. Branches should be created by using grafting short side clusters. Branches should not exceed a length of three sites. Target candidates are chosen at random from the candidate list following from the chosen radius.

7.4.8.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1,
  radius        -> 0
};
```

7.4.9 Rebridge

7.4.9.1 Syntax

```
rebridge        = {
  active        -> boolean,
  frequency     -> integer,
  drivers       -> integer,
  tolerance     -> real,
  n             -> integer,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`drivers`	`integer`	Number of driver sites; valid options are `0`, `1`, and `2`.
`tolerance`	`real`	Move tolerance; currently not in use.
`n`	`integer`	Number of masses available in the simulation; controlled internally: output only.
`accept`	`struct`	Array of accumulative accepted and total trials; output only.

7.4.9.2 Usage

The `rebridge` move performs concerted rotations on single chains to promote accelerated equilibration of long chain polymers.[1]

7.4.9.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1
};
```

1. V.G. Mavrantzas, T.D. Boone, E. Zervopoulou, and D.N. Theodorou, "End-Bridging Monte Carlo: A Fast Algorithm for Atomistic Simulation of Condensed Phases of Long Polymer Chains", *Macromolecules* **1999**, *32*, 5072.

7.4.10 Reptate

7.4.10.1 Syntax

```
reptate = {
  active        -> boolean,
  frequency     -> integer,
  target        -> constant,
  nmin          -> integer,
  n             -> integer,
  accept        -> struct
};
```

Directive	Parameters	Description
active	boolean	Indicates the state of the move; options are `true` or `false`.
frequency	integer	Sets the frequency of the move, which is assumed to be a positive integer.
target	constant	Optional target end group and site; considers all available chain ends when not defined.
nmin	integer	Minimum length of the resulting chain; valid values are values larger or equal than three.
n	integer	Number of masses available in the simulation; controlled internally: output only.
accept	struct	Array of accumulative accepted and total trials; output only.

7.4.10.2 Usage

The `reptate` move promotes chain reptation by taking the end of one chain and moving it to the end of another.

7.4.10.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1,
  nmin          -> 3
};
```

7.4.11 Rotate

7.4.11.1 Syntax

```
rotate  = {
  active        -> boolean,
  frequency     -> integer,
  nmax          -> integer,
  n             -> integer,
  amax          -> real,
  accept        -> struct
};
```

Directive	Parameters	Description
active	boolean	Indicates the state of the move; options are `true` or `false`.
frequency	integer	Sets the frequency of the move, which is assumed to be a positive integer.
nmax	integer	Currently not in use.
n	integer	Number of masses available in the simulation; controlled internally: output only.
amax	real	Array of adapted rotation limits based on simulation progress; internally determined using a feedback loop; output only.
accept	struct	Array of accumulative accepted and total trials; output only.

7.4.11.2 Usage

The `rotate` move selects and rotate chain ends at random. One chain is considered per attempted move. Per attempted on to three sites are included in a random rotation.

7.4.11.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1
};
```

7.4.12 Surface

7.4.12.1 Syntax

```
surface = {
  n             -> integer,
  dmax          -> real,
  accept        -> struct
};
```

Directive	Parameters	Description
n	integer	Number of masses available in the simulation; controlled internally: output only.
dmax	real	Array of adapted displacement limits based on simulation progress; internally determined using a feedback loop; output only.
accept	struct	Array of accumulative accepted and total trials; output only.

7.4.12.2 Usage

The `surface` move represents a subclass of the displacement move and operates on sites that are part of a body representing a surface. The move is automatically invoked when the latter applies. User settings are controlled via the displacement move.

7.4.12.3 Default

Defaults are given by the displacement move.

7.4.13 Temper

7.4.13.1 Syntax

```
temper  = {
  active        -> boolean,
  frequency     -> integer,
  n             -> integer,
  accept        -> struct
};
```

Directive	Parameters	Description
`active`	`boolean`	Indicates the state of the move; options are `true` or `false`.
`frequency`	`integer`	Sets the frequency of the move, which is assumed to be a positive integer.
`n`	`integer`	Number of masses available in the simulation; controlled internally: output only.
`accept`	`struct`	Array of accumulative accepted and total trials; output only.

7.4.13.2 Usage

The `temper` move controls the exchange between multiple replicas as represented by existing systems. The exchange is governed by the per system chosen temperatures.

7.4.13.3 Default

The default is given by

```
sample          = {
  active        -> false,
  frequency     -> 1
};
```

7.5 Port

7.5.1 Syntax

```
port            = {forcefield -> option};
```

Directive	Parameters	Description
`forcefield`	`option`	Sets the ported force field type.

Option	Description
`none`	No force field interpretation; bonded contributions are not included in the ported format.
`auto`	Automated guess of ported format; based on which force field family is currently active; only works correctly when just one force field family is activated.
`boltzmann`	Assumes Boltzmann force fields (see Section 7.12.4 [Boltzmann], page 207).
`cff`	Assumes CFF force fields (see Section 7.12.6 [CFF], page 220).
`charmm`	Assumes CHARMM force fields (see Section 7.12.7 [CHARMM], page 236).
`dpd`	Assumes DPD coarse-grained force fields (see Section 7.12.11 [DPD], page 275).
`gauss`	Assumes Gaussian coarse-grained force fields (see Section 7.12.12 [Gauss], page 284).
`martini`	Assumes MARTINI coarse-grained force fields (see Section 7.12.14 [MARTINI], page 302).
`mie`	Assumes Mie coarse-grained force fields (see ⟨undefined⟩ [mie], page ⟨undefined⟩).
`opls`	Assumes OPLS all-atom and united-atom force fields (see Section 7.12.16 [OPLS], page 319).
`sdk`	Assumes SDK coarse-graine force fields (see Section 7.12.17 [SDK], page 328).
`standard`	Assumes standard force fields (see Section 7.12.19 [Standard], page 341).
`table`	Assumes tabulated force fields (see Section 7.12.20 [Table], page 350).
`trappe`	Assumes TraPPE all-atom and united-atom force fields (see Section 7.12.21 [TraPPE], page 356).
`coarse`	Assumes coarse-grained force fields.
`colloid`	Assumes colloidal force fields.
`fene`	Assumes FENE force fields.

7.5.2 Default

Unless otherwise stated, the default is given by

```
port            = {forcefield -> auto};
```

7.6 Profiles

7.6.1 Syntax

```
profiles        = {
  bond          -> struct,
  density       -> struct,
  force         -> struct,
  mass          -> struct,
  order         -> struct
};
```

Directive	Parameters	Description
`bond`	`struct`	Settings for bond length distributions of all bonds within all systems; no discrimination is made between bond types.
`density`	`struct`	Settings for mass density profiles; no discrimination is made between site masses.
`force`	`struct`	Settings for energy and virial density profiles; resulting binned contributions reflect the interaction of that bin with its surroundings.
`mass`	`struct`	Settings for mass profiles; distinction is made between contributing site masses.
`order`	`struct`	Settings for order profiles; no distinction is made between contributing bond types.

7.6.2 Usage

This variable style describes profiles.

7.6.3 Default

By default, all profiles are deactivated and all entries are zeroed out.

7.7 Region

7.7.1 Syntax

```
region          = {shape -> option, type -> option, mode -> option,
                    center -> vector, h -> voigt[, radius -> vector]}
```

Directive	Parameters	Description
`shape`	`option`	Sets region shape; possibilities are `cuboid` or `spheroid`.
`type`	`option`	Sets region type; possibilities are `relative` or `absolute`.
`mode`	`option`	Sets region mode; possibilities are `hard` or `soft`; used in conjunction with sites growth.
`center`	`vector`	Defines the center of a region.
`h`	`voigt`	Defines the extent of a region as a voigt notation shape (see Section 7.15 [Voigt], page 367).
`radius`	`vector`	Defines the extent of a region as a vector (see Section 7.14 [Vector], page 366); the `radius` is a derived alternative to `h` and translated into `h` upon execution.

7.7.2 Usage

Specifies a region of a certain shape within system simulation cell. Units of the shape center and radius can either be expressed in a relative or an absolute fashion. Relative units express both shape center and radius in units of base vectors $\vec{a}$, $\vec{b}$, and $\vec{c}$ and lengths thereof. Absolute units make use of the simulation-wide settings as expressed under in the 'Units' section (see Section 7.13 [Units], page 364) of the simulation structure. The region's mode influences the growth of sites by either `hard` or `soft` exclusion or inclusion. The `hard` mode applies the region to all sites of inserted clusters, while the `soft` mode only applies the region to the first site of inserted clusters.

7.7.3 Default

Unless otherwise stated, the default is given by

```
region          = {shape -> spheroid, type -> relative, mode -> hard,
                    center -> {0,0,0}, h -> {0,0,0,0,0,0}};
```

7.8 SMILES

7.8.1 Syntax

```
chemistry       = string;
```

7.8.2 Usage

The Simplified Molecular Input Line Entry System (or SMILES for short) is used to describe chemistry as reflected in the definitions of `groups` (see Section 5.17 [Groups], page 117). It follows the syntax as described Daylight Chemical Information Systems, Inc., but is extended to also capture non-periodic system representations as are used in defining coarse-grained systems. The distinction between periodic and non-periodic representations is identified by keywords `atomistic` and `coarse` respectively. In `atomistic` mode, hydrogen completion is implicit and lower character case signifies aromaticity, where allowed characters are `c`, `n`, `o`, `p`, and `s`. A short summary is listed in the table below. Extensions to the original SMILES format are marked with (extension).

Modifier	**Example**	**Description**
`>X`	`>C`	First branch point, connecting to chains in the down direction (extension).
`X<`	`C<`	Second branch point, connecting to chains in the up direction (extension).
`*`	`*`	General branch point, connecting chains (extension).
`~X`	`~C`	Represents any bond (extension).
`-X`	`-C`	Represents a single bond (extension).
`:X`	`:C`	Represents a partial double bond (extension).
`=X`	`=C`	Represents a double bond.
`#X`	`#N`	Represents a triple bond.
`x`	`c`	Lower case indicate aromatics in `atomistic` mode; allowed characters are `c`, `n`, `o`, `p`, and `s`.
`[XX]`	`[He]`	Needed for usage of symbols with more than one character.
`()`	`CC(C)C`	Indicates branching from backbone.
`(X)1`	`(C)3`	Indicates repeating sequences (here `(C)3 = CCC`); note, that `CC(C)3C = CCCCCC` and `CC((C)3)C = CC(CCC)C`; multi-digits are allowed (extension).
`1X`	`2H`	Assignes an aberrant mass to a symbol (here 2).
`X1`	`c1ccccc1`	Indicates a link number to create ring structures (here a benzene ring); duplicate link numbers are allowed.
`%`	`c%11ccccc%11`	Used for multi-digit link numbers.
`\X` or `/X`	`C/C=C/C`	Identifies cis (`C/C=C\C`) or trans (`C/C=C/C`) isomers.
`@`	`[C@H]`	Identifies L- (`[C@@H]`) or R- (`[C@H]`) stereoisomers.
`+1`	`[Na+1]`	Indicates an assigned positive charge; charges can be partial (e.g. 0.5); a `+` with omitted digits indicates an increase by `+1`, e.g. `[Na+]` equals `[Na+1]` or `[Ca++]` represents a calcium atom with charge `+2`.

`-1`	`[Cl-1]`	Indicates an assigned negative charge; charges can be partial (e.g. 0.5); a - with omitted digits indicates an decrease by -1.

7.8.3 Examples

The following table shows a few example chemistry and their corresponding SMILES strings.

Chemistry	**SMILES**
water	`HOH`
cyclohexanone	`O=C1CCCCC1`
dodecane	`(C)12`
	`CCCCCCCCCCCC`
iso-octane	`CC(C)CC(C)(C)C`
diphenylmethane	`c1ccccc1Cc1ccccc1`
Methylene diphenyl 4,4'-diisocyanate	`O=C=Nc1ccc(cc1)Cc2ccc(N=C=O)cc2`
trans-2-butene	`C/C=C/C`
L-tryptophan	`c1ccc2c(c1)c(c[nH]2)C[C@@H](C(=O)O)N`

7.9 Splines

7.9.1 Introduction[1]

In mathematics, a spline is a sufficiently smooth polynomial function that is piecewise-defined, and possesses a high degree of smoothness at the places where the polynomial pieces connect (which are known as knots).[2,3]

In interpolating problems, spline interpolation is often referred to as polynomial interpolation because it yields similar results, even when using low-degree splines, to interpolating with higher degree polynomials while avoiding instability due to Runge's phenomenon. In computer graphics splines are popular curves because of the simplicity of their construction, their ease and accuracy of evaluation, and their capacity to approximate complex shapes through curve fitting and interactive curve design.

The most commonly used splines are cubic spline, i.e., of order 3 – particular, cubic B-spline and cubic Bezier spline. They are common, in particular, in spline interpolation simulating the function of flat splines. The term spline is adopted from the name of a flexible strip of metal commonly used by draftsmen to assist in drawing curved lines.[4]

Splines are curves, which are usually required to be continuous and smooth. Splines are usually defined as piecewise polynomials of degree n with function values and first n-1 derivatives that agree at the points where they join. The abscissa values of the join points are called knots. The term "spline" is also used for polynomials (splines with no knots) and piecewise polynomials with more than one discontinuous derivative. As such, splines with no knots are generally smoother than splines with knots, which are generally smoother than splines with multiple discontinuous derivatives. Splines with few knots are generally smoother than splines with many knots; however, increasing the number of knots usually increases the fit of the spline function to the data. Knots give the curve freedom to bend to more closely follow the data.

It is commonly accepted that the first mathematical reference to splines is the 1946 paper by Schoenberg,[5] which is probably the first place that the word "spline" is used in connection with smooth, piecewise polynomial approximation. However, the ideas have their roots in the aircraft and shipbuilding industries.

7.9.2 Linear Spline

The linear splined applied by EMC follows

$$\Delta x_i = x - x_i,$$
$$S_i(x) = \sum_{j=0}^{1} k_{j,i} \Delta x_i^j,$$
$$k_{0,i} = y_i,$$
$$k_{1,i} = (y_i - y_{i-1})/(x_i - x_{i-1}).$$

Linear splines find their application in cases of sudden large gradients or steps in y data with respect to x.

7.9.3 Cubic Spline

Internally, EMC applies a natural cubic spline by default. The algorithm behind a natural cubic spline is given by

$$\Delta x_i = x - x_i,$$
$$S_i(x) = \sum_{j=0}^{3} k_{j,i} \Delta x_i^j,$$
$$S_i(x_i) = S_{i-1}(x_i) = y_i,$$
$$S_i'(x_i) = S_{i-1}'(x_i),$$
$$S_i''(x_i) = S_{i-1}''(x_i),$$
$$S_i''(x_0) = S_{n-1}''(x_n) = 0,$$

for which the above set of equations defines all constants k needed to describe the full spline function. Cubic splines are useful for capturing the behavior of reasonably well-behaved data.

7.9.4 References

1. Wikipedia on "*Cubic Splines*"
2. K. L. Judd, "*Numerical Methods in Economics*", *MIT Press* **1998**, 225 (ISBN 978-0-262-10071-7).
3. W.-K. Chen, Wai-Kai (2009 "*Feedback, Nonlinear, and Distributed Circuits*", *CRC Press* **2009**, 9-20 (ISBN 978-1-4200-5881-9).
4. M. H. Katz, "*Multivariable Analysis: A Practical Guide for Clinicians and Public Health Researchers*", *Cambridge University Press* **2011**, 82 (ISBN 978-0-521-14107-9).
5. Schoenberg, "*Contributions to the problem of approximation of equidistant data by analytic functions*", *Quart. Appl. Math.* **1946**, *4*, 45-99 and 112-141.

7.10 System Flags

7.10.1 Syntax

```
flag            = {charge -> boolean, map -> boolean, pbc -> boolean};
```

Directive	Parameters	Description
`charge`	`boolean`	Check for charge neutrality; options are `true` or `false`.
`map`	`boolean`	Map configurations into the system box; options are `true` or `false`.
`pbc`	`boolean`	Apply periodic conditions to system geometry; options are `true` or `false`.

7.10.2 Default

Unless otherwise stated, the default is given by

```
flag            = {charge -> true, map -> true, pbc -> true};
```

7.11 Systems

7.11.1 Syntax

```
systems         = {
  n             -> integer,
  properties    -> {
    {
      id        -> constant,
      p         -> real,
      v         -> real,
      t         -> real,
      mass      -> real,
      nclusters -> integer,
      nsites    -> struct,
      plane     -> integer,
      geometry  -> voigt
    },
    ...
  }
};
```

Directive	Parameters	Description
`n`	`integer`	Sets the number of contributing systems to the total simulation.
`properties`		Describes the properties of each individual system.
`id`	`constant`	Identifies the system; refers to the `systems` paragraph in `constants` (see Section 7.1 [Constants], page 179).
`p`	`real`	Defines the system pressure; only used when deformation moves are active.
`v`	`real`	Defines the system volume.
`t`	`real`	Defines the system temperature.
`mass`	`real`	Reflects the total system mass; cannot be altered.
`nclusters`	`integer`	Reflects the number of clusters in this system; cannot be altered.
`nsites`	`struct`	Reflects the number of sites in this system; cannot be altered.
`plane`	`integer`	Reflects the original crystal plane the system was constructed with.
`geometry`	`voigt`	Defines the system geometry (see Section 7.15 [Voigt], page 367).

7.11.2 Usage

This variable style describes the definition of systems.

7.11.3 Default

Unless otherwise stated, the default is given by

```
systems          = {n -> 0};
```

7.12 Types

7.12.1 Syntax

```
types           = {
  merge         -> boolean,
  virial        -> boolean,
  periodic      -> vector,
  neighbor      -> constant,
  stencil       -> constant,
  skin          -> real,
  shake         -> constant
  depth         -> integer,
  mass          -> struct,
  boltzmann     -> struct,
  charmm        -> struct,
  cff           -> struct,
  coarse        -> struct,
  colloid       -> struct,
  coulomb       -> struct,
  dpd           -> struct,
  gauss         -> struct,
  gromacs       -> struct,
  inverse       -> struct,
  martini       -> struct,
  opls          -> struct,
  sdk           -> struct,
  spline        -> struct,
  standard      -> struct,
  table         -> struct,
  trappe        -> struct
};
```

Directive	Parameters	Description
`merge`	`boolean`	Allows for merging force field constants upon input when true; options are `true` or `false`.
`virial`	`boolean`	Describes if virial calculations are included; options are `true` or `false`.
`periodic`	`vector`	Indicate periodicity with a three-element boolean vector with options `true` or `false`.
`neighbor`	`constant`	Describes what kind of neighbor list algorithm is used during pair interaction calculations; options are `sector` or `pair`.
`stencil`	`constant`	Describes the kind of stencil used during pair interaction calculations; options are `standard` or `multi`.

`skin`	`real`	Describes the skin used during pair interaction calculations; the skin is added to the pairwise cutoff.
`shake`	`constant`	Indicates the use of the SHAKE algorithm in subsequent codes (e.g. LAMMPS); valid options are `none`, `auto`, `hydrogen`, `water`, or `all`.
`depth`	`integer`	Maximum depth used for construction of ring structures during typing; allowed values are positive, where a value of 8 works in most ring cases; alternatively, an `auto` keyword allows for checking rings of unknown size; please note, that significant slow down occurs with the latter options for intricate ring systems.
`mass`	`struct`	Describes the site masses.
`boltzmann`	`struct`	Describes Boltzmann force fields (see Section 7.12.4 [Boltzmann], page 207).
`born`	`struct`	Describes Born force fields (see Section 7.12.5 [Born], page 212).
`cff`	`struct`	Describes CFF (Class2) force field families (see Section 7.12.6 [CFF], page 220).
`charmm`	`struct`	Describes CHARMM force fields (see Section 7.12.7 [CHARMM], page 236).
`coarse`	`struct`	Describes multiple coarse-grained force field definitions (see Section 7.12.8 [Coarse], page 247).
`colloid`	`struct`	Describes colloidal force fields (see Section 7.12.9 [Colloid], page 263).
`coulomb`	`struct`	Describes coulombic contribution definitions (see Section 7.12.10 [Coulomb], page 271).
`martini`	`struct`	Describes MARTINI coarse-grained force fields (see Section 7.12.14 [MARTINI], page 302).
`gromacs`	`struct`	Describes GROMACS force fields (see Section 7.12.13 [GROMACS], page 293).
`inverse`	`struct`	Describes settings used in applications of inverse bonded interactions.
`dpd`	`struct`	Describes DPD coarse-grained force fields (see Section 7.12.11 [DPD], page 275).
`gauss`	`struct`	Describes Gaussian coarse-grained force fields (see Section 7.12.12 [Gauss], page 284).
`opls`	`struct`	Describes OPLS force fields (see Section 7.12.16 [OPLS], page 319).
`sdk`	`struct`	Describes SDK force fields (see Section 7.12.17 [SDK], page 328).
`spline`	`struct`	Describes spline-based force fields (see Section 7.12.18 [Spline], page 337).
`standard`	`struct`	Describes standard force fields (see Section 7.12.19 [Standard], page 341).

`table`	`struct`	Describes tabular force fields (see Section 7.12.20 [Table], page 350).
`trappe`	`struct`	Describes tabular force fields (see Section 7.12.21 [TraPPE], page 356).

7.12.2 Usage

This variable style describes types.

7.12.3 Default

Unless otherwise stated, the default is given by

```
types           = {
  merge         -> false,
  virial        -> false,
  periodic      -> {true, true, true},
  neighbor      -> sector,
  stencil       -> standard,
  shake         -> none,
  depth         -> auto,
  skin          -> 0
};
```

By default, all force fields are deactivated.

7.12.4 Boltzmann

7.12.4.1 Syntax

```
boltzmann       = {
  bond          -> struct,
  angle         -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
pair	struct	Pair interaction descriptors

The boltzmann force field uses exponential functions for bonded interactions and tabulated functions for nonbonded interactions. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{boltzmann} = E_{bond} + E_{angle} + E_{pair}$$

for angle, bond and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/nanoparticle/coarse/ for an application.

7.12.4.2 Example

```
variables       = {
  a             -> 1*^-10,
  prefix        -> "~/emc/version/force/polystyrene/smoothed_"
};

simulation      = {
  types         -> {
    merge       -> true,
    boltzmann   -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0 -> m, i1 -> m, coefficients -> {
            a -> 0.015e-10, w -> 0.09e-10, length -> 2.46e-10}},
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0 -> m, i1 -> m, i2 -> m, coefficients -> {
              {a -> 0.140, w -> 14.2, theta -> 147.3},
              {a -> 0.030, w -> 15.5, theta -> 158.0}}}
```

```
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 2,
        data    -> {
          {i0 -> m, i1 -> m, order -> 2, name -> prefix+"m-m.m"}
        }
      }
    }
  }
};
```

7.12.4.3 References

1. T. Spyriouni, C. Tzoumanekas, D. Theodorou, F. Mueller-Plathe, and G. Milano, "*Coarse-Grained and Reverse-Mapped United-Atom Simulations of Long-Chain Atactic Polystyrene Melts: Structure, Thermodynamic Properties, Chain Conformation, and Entanglements*", *Macromolecules* **2007**, *40*, 3876

7.12.4.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      cutoff    -> real,
      order     -> integer,
      name      -> string
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`cutoff`	`real`	Defines the cutoff of the potential; takes the last data file entry when `cutoff <= 0`.
`order`	`integer`	Interpolation order; either 1 for linear interpolation or 2 for cubic splines.
`name`	`string`	Defines the data file name; ./force/polystyrene/ shows examples of the data file format (i.e. {{x, y}, ...}).

The energetic functional form of the pair contributions to the total potential is formed by a linear interpolation or a cubic spline through the provided data.

7.12.4.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      coefficients ->
      {
        {
          a     -> real,
          w     -> real,
          length-> real
        },
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`coef`	`struct`	Describes individual coefficient contributions
`a, w`	`real`	Force constants
`length`	`real`	Equilibrium length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = \sum_{i=1}^{n} a_i/(2\pi w_i)\exp[-2(l - l_{0,i})^2/w_i^2],$$

where l represents the bond length.

7.12.4.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      coefficients ->
      {
        {
          a     -> real,
          w     -> real,
          theta -> real
        },
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to `mass` paragraph in `types`
`coef`	`struct`	Describes individual coefficient contributions
`a, w`	`real`	Force constants
`theta`	`real`	Equilibrium angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = \sum_{i=1}^{n} a_i/(2\pi w_i) \exp[-2(\theta - \theta_{0,i})^2/w_i^2],$$

where θ represents the bond angle.

7.12.5 Born

7.12.5.1 Syntax

```
born            = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The OPLS force field is a compounded force field based on the Born force field.[1−2]. Bonded terms are equivalent to OPLS force fields[3]. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{opls} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See `./examples/field/born/` for applications.

7.12.5.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    born        -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
```

```
      torsion   -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.5.3 References

1. F.G. Fumi and M.P. Tosi, *Ionic sizes and born repulsive parameters in the NaCl-type alkali halides—I: The Huggins-Mayer and Pauling forms*, *J. Phys. Chem. Solids* **1964**, *25*, 31-44.
2. F.G. Fumi and M.P. Tosi, *Ionic sizes and born repulsive parameters in the NaCl-type alkali halides—II: The generalized Huggins-Mayer form*, *J. Phys. Chem. Solids* **1964**, *25*, 31-44.
3. W. L. Jorgensen and J. Tirado-Rives, "*The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.5.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      sigma     -> real,
      rho       -> real,
      a         -> real,
      c         -> real,
      d         -> real,
      core      -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants

`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`sigma`	`real`	Interaction dependent parameter.
`rho`	`real`	Ionic pair dependent paramater.
`a`	`real`	Exponential prefactor.
`c`	`real`	Atractive parameter.
`d`	`real`	Repulsive parameter.
`core`	`real`	Distance at which the potential is treated as a purely repulsive potential; needed to avoid singular behavior at small distances.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = A \exp\left(\frac{\sigma - r_{ij}}{\rho}\right) - \frac{C}{r_{ij}^6} + \frac{D}{r_{ij}^8},$$

for $r_{ij} > r_{core}$, where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.5.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.5.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.5.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} 1/2\ k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.5.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.6 CFF

7.12.6.1 Syntax

```
cff                    = {
  pair                 -> struct,
  bond                 -> struct,
  angle                -> struct,
  bond_bond            -> struct,
  bond_angle           -> struct,
  torsion              -> struct,
  end_bond_torsion     -> struct,
  middle_bond_torsion  -> struct,
  bond_bond_13         -> struct,
  angle_torsion        -> struct,
  angle_angle_torsion  -> struct,
  improper             -> struct,
  angle_angle          -> struct
};
```

Directive	Parameters	Description
pair	struct	Pair interaction descriptors
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
bond_bond	struct	Bond-bond interaction descriptors
bond_angle	struct	Bond-angle interaction descriptors
torsion	struct	Torsion interaction decriptors
end_bond_torsion	struct	End bond torsion interaction decriptors
middle_bond_torsion	struct	Middle bond torsion interaction decriptors
bond_bond_13	struct	Bond-bond-13 interaction decriptors
angle_torsion	struct	Angle-torsion interaction decriptors
angle_angle_torsion	struct	Angle-angle-torsion interaction decriptors
improper	struct	Improper interaction decriptors
angle_angle	struct	Angle-angle interaction decriptors

The CFF force field, as described by Sun,[1] uses a combination of standard and cross-coupled bonded interactions combined with Lennard-Jones 6-9 nonbonded interactions. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{cff} = E_{bond} + E_{angles} + E_{torsions} + E_{impropers} + E_{pair}$$
$$E_{angles} = E_{angle} + E_{bond-bond} + E_{bond-angle}$$
$$E_{torsions} = E_{torsion} + E_{end-bond-torsion} + E_{middle-bond-torsion} + E_{bond-bond-13} + E_{angle-torsion} + E_{angle-angle-torsion}$$
$$E_{impropers} = E_{improper} + E_{angle-angle}$$

for bond, angle, torsion, improper, and pair contributions respectively. The following paragraphs describe each contribution in detail. Note, that improper contributions are calculated differently (see Improper). Force field typing is available for the PCFF force field.

7.12.6.2 Examples

The following example creates topology for water and types these sites with the PCFF force field as included with EMC,

```
field           = {
  mode          -> cff,
  name          -> {$root+"pcff/pcff_templates.dat", $root+"pcff/pcff.frc"
};

groups          = {
  group         -> {id -> water, chemistry -> "O"}
}

clusters        = {
  cluster       -> {id -> water, system -> main, group -> water, n -> 1000}
};

field           = {
  mode          -> apply
};
```

Application of the `field` scripting command saves the need for manual force field typing (see Section 5.11 [Field], page 106). However, CFF-type force field alterations or additions can also be entered directly through

```
simulation      = {
  types         -> {
    merge       -> true,
    cff -> {active -> true,
      angle     -> {
        active  -> true, n -> 1, data ->
          {i0   -> hw, i1 -> o*, i2 -> hw, theta -> 103.7, k -> {49.84,
            -11.6, -8}}},
      angle_angle -> {
        active  -> true, n -> 0},
      angle_angle_torsion -> {
```

```
        active  -> true, n -> 0},
      angle_torsion -> {
        active  -> true, n -> 0},
      bond      -> {
        active  -> true, n -> 1, data ->
          {i0   -> o*, i1 -> hw, l -> 0.97, k -> {563.28, -1428.22,
            1902.12}}},
      bond_angle -> {
        active  -> true, n -> 1, data ->
          {i0   -> hw, i1 -> o*, i2 -> hw, k -> {22.35, 22.35}}},
      bond_bond -> {
        active  -> true, n -> 1, data ->
          {i0 -> hw, i1 -> o*, i2 -> hw, k -> -9.5}},
      bond_bond_13 -> {
        active  -> true, n -> 0},
      end_bond_torsion -> {
        active  -> true, n -> 0},
      improper  -> {
        active  -> true, n -> 0},
      middle_bond_torsion -> {
        active  -> true, n -> 0},
      pair -> {
        active  -> true, mix -> sixth, shift -> false, coulomb -> cut,
        nbonded -> 2, mode -> repulsive, cutoff -> 9.5, n -> 3, data -> {
          {i0   -> o*, i1 -> o*, epsilon -> 0.274, sigma -> 3.608},
          {i0   -> o*, i1 -> hw, epsilon -> 0.00336154972409,
            sigma -> 3.21478799199},
          {i0   -> hw, i1 -> hw, epsilon -> 0.013, sigma -> 1.098}}},
      torsion   -> {
        active  -> true, n -> 0}},
  }
};
```

The above definition would result from application of the PCFF force field.

7.12.6.3 References

1. H. Sun, "*COMPASS: An ab Initio Force-Field Optimized for Condensed-Phase Applications - Overview with Details on Alkane and Benzene Compounds*", *J. Phys. Chem. B* **1998**, *102*, 7338.

7.12.6.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants.
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-9 potential.
`sigma`	`real`	Site size; the location of the potential well.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-9 Lennard-Jones potential,

$$E_{pair} = \epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[2\left(\frac{\sigma}{r_{ij}}\right)^3 - 3\right],$$

where r_{ij} represents the distance between site i and site j. The nonbond interactions, which include Lennard-Jones and coulombic interactions, encompass all pairs between atoms which are separated by two or more bonded atoms, i.e. `nbonded` = 2. The default mixing rule is sixth power (`sixth`).

7.12.6.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      l         -> real,
      k         -> {real, real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`l`	`real`	Equilibrium length
`k`	`real`	Defines up to three bond constants

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = \sum_{i=1}^{3} k_i (l - l_{0,i})^{i+1},$$

where l represents the bond length of bond $\{i_0, i_1\}$.

7.12.6.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id
      theta     -> real,
      k         -> {real, real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to `mass` paragraph in `types`
`theta`	`real`	Equilibrium angle
`k`	`real`	Defines up to three angle constants

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = \sum_{i=1}^{3} k_i(\theta - \theta_{0,i})^{i+1},$$

where θ represents the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$.

7.12.6.7 Bond-Bond

```
bond_bond       = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id
      k         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to `mass` paragraph in `types`
`k`	`real`	Defines the bond-bond cross term constant

The energetic functional form of the bond-bond cross-coupling contributions to the total potential is described by

$$E_{bond-bond} = k(l_1 - l_{0,1})(l_2 - l_{0,2}),$$

where l_1 represents the length of bond $\{i_0, i_1\}$ with equilibrium value $l_{0,1}$ and l_2 the length of bond $\{i_1, i_2\}$ with equilibrium value $l_{0,2}$. Equilibrium values reference the bond contributions.

7.12.6.8 Bond-Angle

```
bond_angle      = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id
      k         -> {real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to `mass` paragraph in `types`
`k`	`real`	Defines the bond-angle cross term constants

The energetic functional form of the bond-angle cross-coupling contributions to the total potential is described by

$$E_{bond-angle} = (k_1(l_1 - l_{0,1}) + k_2(l_2 - l_{0,2}))(\theta - \theta_{0,i}),$$

where l_1 represents the length of bond $\{i_0, i_1\}$ with equilibrium value $l_{0,1}$, l_2 the length of bond $\{i_1, i_2\}$ with equilibrium value $l_{0,2}$, and θ the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$. Equilibrium values reference both bond and angle contributions.

7.12.6.9 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {k      -> real, n -> integer, delta -> real},
        {k      -> real, n -> integer, delta -> real},
        {k      -> real, n -> integer, delta -> real}
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`
`coefficients`	`struct`	List of up to three sets of coefficients
`k`	`real`	Torsion constants
`n`	`integer`	Torsion prefactor
`delta`	`real`	Torsion offset

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{3} k_i(1 - cos[n_i\phi - \delta]),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$.

7.12.6.10 End-Bond-Torsion

```
end_bond_torsion = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> {real, real, real, real, real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`
`k`	`real`	Three left (k_1 through k_3) and three right (k_4 through k_6) interaction constants

The energetic functional form of the end-bond-torsion cross-coupling contributions to the total potential is described by

$$E_{end-bond-torsion} = \sum_{i=1}^{3}(k_i(l_1 - l_{0,1}) + k_{i+3}(l_2 + l_{0,2}))\cos[i\phi],$$

where l_1 represents the length of bond $\{i_0, i_1\}$ with equilibrium value $l_{0,1}$, l_2 represents the length of bond $\{i_2, i_3\}$ with equilibrium value $l_{0,2}$, and ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$.

7.12.6.11 Middle-Bond-Torsion

```
middle_bond_torsion = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> {real, real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`
`k`	`real`	Three (k_1 through k_3) interaction constants

The energetic functional form of the middle-bond-torsion cross-coupling contributions to the total potential is described by

$$E_{end-bond-torsion} = \sum_{i=1}^{3} k_i(l_1 - l_{0,1}) \cos[i\phi],$$

where l_1 represents the length of bond $\{i_1, i_2\}$ with equilibrium value $l_{0,1}$ and ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$.

7.12.6.12 Bond-Bond-13

```
bond_bond_13    = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants.
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`.
`k`	`real`	Bond-Bond-13 interaction constant.

The energetic functional form of the bond-bond-13 cross-coupling contributions to the total potential is described by

$$E_{bond-bond-13} = k_{bond-bond-13}(l_1 - l_{0,1})(l_2 - l_{0,2}),$$

where l_1 represents the length of bond $\{i_0, i_1\}$ with equilibrium value $l_{0,1}$, l_2 represents the length of bond $\{i_2, i_3\}$ with equilibrium value $l_{0,2}$.

7.12.6.13 Angle-Torsion

```
angle_torsion           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> {real, real, real, real, real, real}
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants.
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`.
`k`	`real`	Six (k_1 through k_6) interaction constants.

The energetic functional form of the angle-torsion cross-coupling contributions to the total potential is described by

$$E_{angle-torsion} =, \sum_{i=1}^{2}(\theta_i - \theta_{0,i}) \sum_{j=1}^{3} k_{3(i-1)+j} \cos[j\phi],$$

where θ_1 the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, θ_2 the angle between bonds $\{i_1, i_2\}$ and $\{i_2, i_3\}$, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$.

7.12.6.14 Angle-Angle-Torsion

```
angle_torsion           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`

The energetic functional form of the angle-angle-torsion cross-coupling contributions to the total potential is described by

$$E_{angle-angle-torsion} =, k_{angle-angle-torsion}((\theta_1 - \theta_{0,1})((\theta_2 - \theta_{0,2})\cos[\phi],$$

where θ_1 the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, θ_2 the angle between bonds $\{i_1, i_2\}$ and $\{i_2, i_3\}$, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$.

7.12.6.15 Improper

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. EMC represents site i_0 as the central site, as opposed to some representations, for which i_1 is the central site. Note, that each improper contributes to the total improper energy individually. Note, that Discover and LAMMPS use the average of the three contributing angles ψ and calculate the improper energy accordingly. EMC, however, uses the average of the energy for all three contributing factors.

7.12.7 CHARMM

7.12.7.1 Syntax

```
charmm          = {
  pair          -> struct,
  bond          -> struct,
  angle         -> struct,
  urey          -> struct,
  torsion       -> struct,
  pair14        -> struct,
  improper      -> struct
};
```

Directive	Parameters	Description
pair	struct	Pair interaction descriptors
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
urey	struct	Urey 1-3 interaction decriptors
torsion	struct	Torsion interaction decriptors
pair14	struct	Pair 1-4 interaction descriptors
improper	struct	Improper interaction descriptors

The CHARMM force field as included in EMC is based on the CHARMM description as defined by MacKerrell et al.[1] EMC provides force field files, which are derived from the original CHARMM force field files, but are governed by typing rules, rather than residue templates, as is the case in the original definition. An included script (`./scripts/charmm.pl`) derives typing rules from the original CHARMM force field files in an automated fashion. The force field files are found in `./field/charmm`. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{charmm} = E_{bond} + E_{angle} + E_{urey} + E_{torsion} + E_{improper} + E_{pair14} + E_{pair}$$

for bond, angle, urey, torsion, pair14, and pair contributions respectively. The following paragraphs describe each contribution in detail. See `./examples/field/charmm/` for applications.

7.12.7.2 Examples

The following example creates topology for water and types these sites with the CHARMM force field as included with EMC,

```
field           = {
  mode          -> charmm,
  name          -> {$root+"field/charmm/c32b1/all27_prot_lipid.prm",
                    $root+"field/charmm/c32b1/all27_prot_lipid.top"}
};
groups          = {
```

```
  group         -> {id -> water, chemistry -> "O"}
}

clusters        = {
  cluster       -> {id -> water, system -> main, group -> water, n -> 1000}
};

field           = {
  mode          -> apply
};
```

Application of the `field` scripting command saves the need for manual force field typing (see Section 5.11 [Field], page 106). However, CHARMM force field alterations or additions can also be entered directly through

```
simulation      = {
  types         -> {
    merge       -> true,
    charmm      -> {active -> true,
      angle     -> {active -> true, n -> 1, data ->
        {i0     -> HT, i1 -> OT, i2 -> HT, k -> 55, theta -> 104.52}},
      bond      -> {active -> true, n -> 1, data ->
        {i0     -> OT, i1 -> HT, k -> 450, l -> 0.9572}},
      improper  -> {active -> true, n -> 0},
      pair      -> {active -> true, mix -> berthelot, shift -> false,
        coulomb -> none, nbonded -> 3, mode -> individual, inner -> 0,
        cutoff  -> 9.5, n -> 3, data -> {
          {i0   -> OT, i1 -> OT, epsilon -> 0.1521, sigma -> 3.15057422683},
          {i0   -> OT, i1 -> HT, epsilon -> 0.0836456812992,
            sigma -> 1.77529387564},
          {i0   -> HT, i1 -> HT, epsilon -> 0.046,
            sigma -> 0.400013524445}}},
      pair14    -> {active -> true, n -> 3, data -> {
          {i0   -> OT, i1 -> OT, epsilon -> 0.1521, sigma -> 3.15057422683},
          {i0   -> OT, i1 -> HT, epsilon -> 0.0836456812992,
            sigma -> 1.77529387564},
          {i0   -> HT, i1 -> HT, epsilon -> 0.046,
            sigma -> 0.400013524445}}},
      torsion   -> {active -> true, n -> 0},
      urey      -> {active -> true, n -> 1, data ->
        {i0     -> HT, i1 -> OT, i2 -> HT, k -> 0, l -> 0}}},
  }
};
```

7.12.7.3 References

1. A. D. MacKerrell, Jr. et al., "*All-Atom Empirical Potential for Molecular Modeling*

and Dynamics Studies of Proteins", *J. Phys. Chem. B* **1998**, *102*, 3586-3616.

7.12.7.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  inner         -> real,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      inner     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`inner`	`real`	Sets global inner start of cut off for `global` mode.
`cutoff`	`real`	Sets global cut off for `global` mode.
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`in which case the entered entries are merged with already existing ones.
`data`	`struct`	Summary of interaction constants.

`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.
`inner`	`real`	Sets pairwise inner start of cut off; can be omitted upon global *inner* definition.
`cutoff`	`real`	Sets pairwise cut off; can be omitted upon global $cutoff$ definition.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between sites i and j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.7.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.7.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.7.7 Urey

```
urey            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Urey bond spring constants
`l`	`real`	Equilibrium urey bond length

The energetic functional form of the Urey-Bradley contributions to the total potential is described by

$$E_{urey} = k_{urey}(l - l_0)^2,$$

where k_{urey} represents a spring constant, l the urey bond length of bond $\{i_0, i_2\}$, and l_0 the equilibrium bond length.

7.12.7.8 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \cos(n_i\phi - \delta_i)),$$

where k_i represents a set of torsion constants, δ_i a torsion offset, n_i the torsion pre-factor, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. The maximum allowed value of n_i is 6.

7.12.7.9 Pair14

```
pair            = {
  active        -> boolean,
  mix           -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`in which case the entered entries are merged with already existing ones.
`data`	`struct`	Summary of interaction constants.
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{l}\right)^6 \left[\left(\frac{\sigma}{l}\right)^6 - 1\right],$$

where l represents the distance between sites $i0$ and $i1$.

7.12.7.10 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.8 Coarse

7.12.8.1 Syntax

```
coarse          = {
  lj            -> struct,
  repulsive     -> struct,
  sphere        -> struct,
  colloid       -> struct,
  dpd           -> struct,
  charge        -> struct,
  fene          -> struct,
  angle         -> struct
};
```

Directive	Parameters	Description
`lj`	`struct`	Hard sphere core with Lennard-Jones shell interaction descriptors
`repulsive`	`struct`	Hard sphere core with r^-6 repulsive shell interaction descriptors
`sphere`	`struct`	Coarse-grained sphere interaction descriptors (Girifalco[1])
`colloid`	`struct`	Colloidal interaction descriptors (Everaers and Ejtehadi[2])
`dpd`	`struct`	DPD interaction descriptors (Groot and Warren[3])
`charge`	`struct`	Coarse-grained charge interaction decriptors (internal form)
`fene`	`struct`	FENE interaction descriptors (Kremer and Grest[4])
`angle`	`struct`	Angle interaction descriptors

The coarse force field represents a set of coarse-grained interations functions. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{coarse} = E_{LJ} + E_{repulsive} + E_{sphere} + E_{colloid} + E_{DPD} + E_{charge} + E_{FENE} + E_{angle}$$

for hard core Lennard-Jones, hard core repulsive, colloidal, coarse-grained sphere angle, DPD, charge, FENE, and angle contributions respectively. The following paragraphs describe each contribution in detail. Typically, all conributions are not used simulataneously. Note, that the DPD contributions are included for backwards compatibility. Use the DPD force field when typing and transferring to e.g. LAMMPS. See ./examples/nanoparticle/coarse/ for applications.

7.12.8.2 Examples

```
variables       = {
};

simulation      = {
```

```
  types         -> {
    merge       -> true,
    coarse      -> {
      dpd       -> {
        active  -> true,
        data    -> {
        }
      }
    }
  }
};

simulation      = {
  types         -> {
    merge       -> true,
    coarse      -> {
      colloid   -> {
        active  -> true,
        data    -> {
        }
      }
    }
  }
};
```

7.12.8.3 References

1. L. A. Girifalco, "*Molecular Properties of C_{60} in the Gas and Solid Phases*" *J. Chem. Phys.* **1992**, *96*, 858-861.
2. R. Everaers and M. R. Ejtehadi, "*Interaction potentials for soft and hard ellipsoids*", *Phys. Rev. E.* **2003**, *67*, 041710.
3. R. D. Groot and P. B. Warren, "*Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation*", *J. Chem. Phys.* **1997**, *107*, 4423-4435.
4. K. Kremer and G. S. Grest, *J. Chem. Phys.* **1990**, *92*, 5057.

7.12.8.4 LJ

```
lj              = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      core      -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential
`sigma`	`real`	Site size; the point where the potential equals zero

`core`	`real`	Site hard core size
`cutoff`	`real`	Sets the local pairwise cut off

The energetic functional form of the pair contributions to the total potential is described by a hard core, extended with a 6-12 Lennard-Jones potential,

$$E_{LJ} = 4\epsilon \left(\frac{\sigma}{r_{ij}-d_{core}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}-d_{core}}\right)^6 - 1\right] \quad \text{for} \quad r_{ij} \geq d_{core},$$

where r_{ij} represents the distance between site i and site j, and d_{core} its core size.

7.12.8.5 Repulsive

```
repulsive       = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      core      -> real
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential
`sigma`	`real`	Site size; the point where the potential equals zero

`core` `real` Site hard core size

The energetic functional form of the pair contributions to the total potential is described by a hard core, extended with a repulsive potential,

$$E_{repulsive} = 4\epsilon \left(\frac{\sigma}{r_{ij} - d_{core}} \right)^{12} \quad \text{for} \quad r_{ij} \geq d_{core},$$

where r_{ij} represents the distance between site i and site j, and d_{core} its core size.

7.12.8.6 Sphere

```
lj              = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      d         -> real,
      n         -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential
`d`	`real`	Sets the diameter of both particles
`n`	`real`	Sets the number of contributing Lennard-Jones beads to each spherical FCC particle

`cutoff` `real` Sets the local pairwise cut off

The energetic functional form of an integration of two interacting spherical FCC lattices consisting of n Lennard-Jones particles,

$$E_{LJ} = 4\,\epsilon\,n^2\left(\frac{\sigma}{d}\right)^6\left[\frac{1}{12}\left(\frac{1}{s(s-1)^3}+\frac{1}{s(s+1)^3}-\frac{2}{s^4}\right)+\frac{1}{90}\left(\frac{\sigma}{d}\right)^6\left(\frac{1}{s(s-1)^9}+\frac{1}{s(s+1)^9}-\frac{2}{s^{10}}\right)\right],$$

where ϵ and σ are the standard Lennard-Jones constants, $s = r_{ij}/d$, and r_{ij} represents the distance between the particle centers.

7.12.8.7 Colloid

```
colloid         = {
  active        -> boolean,
  nbonded       -> integer,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      A         -> real,
      sigma     -> real,
      d1        -> real,
      d2        -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`A`	`real`	Pairwise Hamaker constants
`sigma`	`real`	Pairwise Lennard-Jones sigma
`d1, d2`	`real`	Respective colloidal particle diameter
`cutoff`	`real`	Pairwise potential cutoff

The energetic functional form of the pair contributions to the total potential, as described by Everaers and Ejtehadi, is formed by the sum of attractive and repulsive energy:

$$E = E_A + E_R$$

Attractive energy is expressed by

$$E_A = -\frac{A}{6}\left[\frac{2d_i d_j}{r_{ij}^2-(d_i+d_j)^2} + \frac{2d_i d_j}{r_{ij}^2-(d_i-d_j)^2} + \ln\left(\frac{r_{ij}^2-(d_i+d_j)^2}{r_{ij}^2-(d_i-d_j)^2}\right)\right],$$

where A is referred to as Hamaker's constant, d_i and d_j represent the diameter sites i and j, and r_{ij} represents the distance between sites. Repulsive energy is expressed by

$$E_R = \frac{A}{37800}\frac{\sigma^6}{r_{ij}}\left[\frac{r_{ij}^2-7r_{ij}(d_i+d_j)+6\left(d_i^2+7d_id_j+d_j^2\right)}{(r_{ij}-d_i-d_j)^7}+\frac{r_{ij}^2+7r_{ij}(d_i+d_j)+6\left(d_i^2+7d_id_j+d_j^2\right)}{(r_{ij}+d_i+d_j)^7}\right.$$
$$\left.-\frac{r_{ij}^2+7r_{ij}(d_i-d_j)+6\left(d_i^2-7d_id_j+d_j^2\right)}{(r_{ij}+d_i-d_j)^7}-\frac{r_{ij}^2-7r_{ij}(d_i-d_j)+6\left(d_i^2-7d_id_j+d_j^2\right)}{(r_{ij}-d_i+d_j)^7}\right],$$

where σ represents the Lennard-Jones (LJ) constant sigma. Hamaker's constant A in LJ units is given by $A = 4\pi\epsilon_{LJ}\left(\rho\sigma^3\right)^2$, with ρ representing reduced density.

7.12.8.8 DPD

```
pair            = {
  active        -> boolean,
  mix           -> option,
  nbonded       -> integer,
  mode          -> option,
  cutoff        -> real,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      a         -> real,
      gamma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`cutoff`	`real`	Sets global cut off for `global` mode
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`a`	`real`	Force interaction constant.
`gamma`	`real`	Force damping constant.
`cutoff`	`real`	Site size; the point where the potential equals zero.

The energetic functional form of pair contributions to the total soft potential is described by a repulsive Hertzian spring,

$$E_{pair} = 1/2\ a\ r_{ij,cutoff}\left(1 - r_{ij}/r_{ij,cutoff}\right)^2, \quad r_{ij} < r_{ij,cutoff}$$

where a represents the pairwise interaction paramater, r_{ij} the distance between site i and site j, and $r_{ij,cutoff}$ the cutoff distance between these two sites.

7.12.8.9 Charge

```
charge          = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      a         -> real,
      kappa     -> real,
      d1        -> real,
      d2        -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`a`	`real`	Model constant
`kappa`	`real`	Electrostatic screening factor
`d1`	`real`	Sets the diameter of particle `i0`
`d2`	`real`	Sets the diameter of particle `i1`
`cutoff`	`real`	Sets the local pairwise cut off

The energetic functional form of the charge contributions to the total potential, which is based on a Yukawa potential. It is described by

$$E_{charge} = A \ln\left(1 + e^{-\kappa H}\right)\left(4 - 2H/d_1 - 2H/d_2\right),$$

where d_1 and d_2 represent particle diameters, $H = r_{ij} - \frac{1}{2}(d_1 + d_2)$, and A the electrostatic constant, which can be expressed in terms of surface potentials ψ_1 and ψ_2,

$$A = \pi\ \epsilon_0\ \epsilon_r\ (\psi_1 + \psi_2)^2 d_1 d_2/(d_1 + d_2)/8,$$

assuming, that both surface potentials are approximately the same.

7.12.8.10 FENE

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = -\tfrac{1}{2}\ k\ l_0^2\ \ln\left[1 - (l/l_0)^2\right],$$

where k represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.8.11 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`

The energetic functional form of the bond contributions to the total potential is described by

$$E_{angle} =$$

where

7.12.9 Colloid

7.12.9.1 Syntax

```
colloid         = {
  pair          -> struct,
  charge        -> struct,
  bond          -> struct,
  angle         -> struct
};
```

Directive	Parameters	Description
pair	struct	Pairwise colloidal interaction descriptors
charge	struct	Pairwise charge interaction descriptors
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors

The colloid force field is a compounded force field, that uses Hamaker and charge screening interactions. Its functional form is a summation of nonbonded and bonded interactions given by

$$E_{colloid} = E_{pair} + E_{charge} + E_{bond} + E_{angle}$$

for pair, charge, bond, and angle contributions respectively. The following paragraphs describe each contribution in detail.

7.12.9.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    standard    -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 400, theta -> 110}
        }
      },
      pair      -> {
        active  -> true,
        inner   -> 1.0001,
        outer   -> 1.25,
```

```
        nbonded -> 2,
        data    -> {
          {i0   -> a, i1 -> a,
            A   -> 1e-21, d1 -> 1e-7, d2 -> 1e-7, cutoff -> 2.5e-7},
          {i0   -> a, i1 -> b,
            A   -> 1.1e-21, d1 -> 1e-7, d2 -> 2e-7, cutoff -> 3.75e-7},
          {i0   -> a, i1 -> c,
            A   -> 1.2e-21, d1 -> 1e-7, d2 -> 3e-7, cutoff -> 5e-7},
          {i0   -> b, i1 -> b,
            A   -> 1.2e-21, d1 -> 2e-7, d2 -> 2e-7, cutoff -> 5e-7},
          {i0   -> b, i1 -> c,
            A   -> 1.3e-21, d1 -> 2e-7, d2 -> 3e-7, cutoff -> 6.25e-7},
          {i0   -> c, i1 -> c,
            A   -> 1.4e-21, d1 -> 3e-7, d2 -> 3e-7, cutoff -> 7.5e-7}
        }
      }
    }
  }
};
```

7.12.9.3 References

1. R. Everaers and M. R. Ejtehadi, "*Interaction potentials for soft and hard ellipsoids*", *Phys. Rev. E.* **2003**, *67*, 041710.

7.12.9.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  n             -> integer,
  inner         -> real,
  outer         -> real,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      A         -> real,
      sigma     -> real,
      d1        -> real,
      d2        -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`inner`	`real`	Fraction of diameter defining the inner interaction rim; must be a number larger than 1.
`outer`	`real`	Fraction of diameter defining the outer interaction rim; must be a number larger than 1.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`A`	`real`	Pairwise Hamaker constants
`sigma`	`real`	Pairwise Lennard-Jones sigma
`d1, d2`	`real`	Respective colloidal particle diameter
`cutoff`	`real`	Pairwise potential cutoff

The energetic functional form of the pair contributions to the total potential, as described by Everaers and Ejtehadi, is formed by the sum of attractive and repulsive energy:

$$E = E_A + E_R$$

Attractive energy is expressed by

$$E_A = -\frac{A}{6}\left[\frac{2d_i d_j}{r_{ij}^2-(d_i+d_j)^2}+\frac{2d_i d_j}{r_{ij}^2-(d_i-d_j)^2}+\ln\left(\frac{r_{ij}^2-(d_i+d_j)^2}{r_{ij}^2-(d_i-d_j)^2}\right)\right],$$

where A is referred to as Hamaker's constant, d_i and d_j represent the diameter sites i and j, and r_{ij} represents the distance between sites. Repulsive energy is expressed by

$$E_R = \frac{A}{37800}\frac{\sigma^6}{r_{ij}}\left[\frac{r_{ij}^2-7r_{ij}(d_i+d_j)+6\left(d_i^2+7d_i d_j+d_j^2\right)}{(r_{ij}-d_i-d_j)^7}+\frac{r_{ij}^2+7r_{ij}(d_i+d_j)+6\left(d_i^2+7d_i d_j+d_j^2\right)}{(r_{ij}+d_i+d_j)^7}\right.$$
$$\left.-\frac{r_{ij}^2+7r_{ij}(d_i-d_j)+6\left(d_i^2-7d_i d_j+d_j^2\right)}{(r_{ij}+d_i-d_j)^7}-\frac{r_{ij}^2-7r_{ij}(d_i-d_j)+6\left(d_i^2-7d_i d_j+d_j^2\right)}{(r_{ij}-d_i+d_j)^7}\right],$$

where σ represents the Lennard-Jones (LJ) constant sigma. Hamaker's constant A in LJ units is given by $A = 4\pi\epsilon_{LJ}\left(\rho\sigma^3\right)^2$, with ρ representing reduced density.

7.12.9.5 Charge

```
charge          = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      a         -> real,
      kappa     -> real,
      d1        -> real,
      d2        -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`a`	`real`	Model constant
`kappa`	`real`	Electrostatic screening factor
`d1`	`real`	Sets the diameter of particle `i0`
`d2`	`real`	Sets the diameter of particle `i1`
`cutoff`	`real`	Sets the local pairwise cut off

The energetic functional form of the charge contributions to the total potential, which is based on a Yukawa potential. It is described by

$$E_{charge} = A \ln\left(1 + e^{-\kappa H}\right)\left(4 - 2H/d_1 - 2H/d_2\right),$$

where d_1 and d_2 represent particle diameters, $H = r_{ij} - \frac{1}{2}(d_1 + d_2)$, and A the electrostatic constant, which can be expressed in terms of surface potentials ψ_1 and ψ_2,

$$A = \pi\ \epsilon_0\ \epsilon_r\ (\psi_1 + \psi_2)^2 d_1 d_2/(d_1 + d_2)/8,$$

assuming, that both surface potentials are approximately the same.

7.12.9.6 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.9.7 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.10 Coulomb

7.12.10.1 Syntax

```
coulomb         = {
  charge        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
charge	struct	Diffuse coarse-grained charge interaction descriptors
pair	struct	Pair interaction descriptors

The coulomb force field is a collection of electrostatic interactions. Its functional form is a summation of nonbonded interactions given by

$$E_{coulomb} = E_{pair} + E_{charge}$$

for charge and pair contributions respectively. The following paragraphs describe each contribution in detail.

7.12.10.2 Examples

```
variables       = {
};

simulation      = {
  types         -> {
    merge       -> true,
    coulomb     -> {
      charge    -> {
        active  -> true,
        data    -> {
        }
      }
    }
  }
};

simulation      = {
  types         -> {
    merge       -> true,
    coulomb     -> {
      pair      -> {
        active  -> true,
        data    -> {
        }
      }
```

```
    }
  }
};
```

7.12.10.3 References

1. R. D. Groot, "*Electrostatic interactions in dissipative particle dynamics - simulation of polyelectrolytes and anionic surfactants*", *J. Chem. Phys.* **2003**, *118*, 11265-11277.
2. M. Gonzalez-Melchor, E. Mayoral, M. E. Valsquez, J. Alejandre, "*Electrostatic interactions in dissipative particle dynamics using the Ewald sums*", *J. Chem. Phys.* **2006**, *125*, 224107.

7.12.10.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  cutoff        -> real
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`

The energetic functional form of the bond contributions to the total potential is described by

$$E_{pair} = \frac{q_i q_j}{4\pi\epsilon_0\epsilon_r r_{ij}},$$

where q_i and q_j represent the point charges of sites i and j respectively, ϵ_0 represents the dielectric permittivity, ϵ_r the dielectric constant, and r_{ij} the distance between site i and site j. Charges are set per site.

7.12.10.5 Charge

```
charge          = {
  active        -> boolean,
  nbonded       -> integer,
  k             -> real,
  cutoff        -> real
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`

The distrubution of charges in a coarse-grained system cannot be described by simple point charges only. Instead, a spatial distribution following from the Poisson-Boltzmann equation (PBE) should be used, as decribed by Groot [Groot 2003]. A first approximation of a two-body approach for the solution to the PBE was given by Gonzalez-Melchor et al. [Gonzalez-Melchor 2006] as a function of exponentials. This solution, however, prooved to inadequately describe the two-body solution. We propose s better approximation, which is based on an error function and is given by

$$E_{charge} = \frac{\mathrm{erf}[kr_{ij}]}{r_{ij}},$$

where k represents a force constant and r_{ij} represents the distance between site i and site j. Charges are set per site. The advantage of this function is its easy integration in already existing standard point-charge descriptions, thus allowing for representation of long-range contributions by Ewald summations without change of already existing implementations.

7.12.11 DPD

7.12.11.1 Syntax

```
dpd             = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair	struct	Pair interaction descriptors

The Dissipative Particle Dynamics (DPD) force field - developed by Groot and Warren - is a compounded force field, that finds its application in mesoscale simulations.[1] Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{dpd} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, improper, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/dpd/ for applications.

7.12.11.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    dpd         -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 20, l -> 1}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 5, theta -> 180}
        }
      },
      torsion   -> {
        active  -> true,
```

```
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 1,
        data    -> {
          {i0   -> a, i1 -> a, a -> 25.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> a, i1 -> b, a -> 26.2, gamma -> 4.5, cutoff -> 1},
          {i0   -> a, i1 -> c, a -> 22.1, gamma -> 4.5, cutoff -> 1},
          {i0   -> a, i1 -> d, a -> 20.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> b, i1 -> b, a -> 25.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> b, i1 -> c, a -> 28.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> b, i1 -> d, a -> 29.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> c, i1 -> c, a -> 25.0, gamma -> 4.5, cutoff -> 1},
          {i0   -> c, i1 -> d, a -> 21.3, gamma -> 4.5, cutoff -> 1},
          {i0   -> d, i1 -> d, a -> 25.0, gamma -> 4.5, cutoff -> 1}
        }
      }
    }
  }
};
```

7.12.11.3 References

1. R.D. Groot and P.B. Warren "*Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation*", *J. Chem. Phys.* **1997**, *107*, 4423-4435.

7.12.11.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mode          -> option,
  mix           -> option,
  cutoff        -> real,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      a         -> real,
      gamma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`cutoff`	`real`	Sets global cut off for `global` mode
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`a`	`real`	Force interaction constant.
`gamma`	`real`	Force damping constant.
`cutoff`	`real`	Site size; the point where the potential equals zero.

The energetic functional form of pair contributions to the total soft potential is described by a repulsive Hertzian spring,

$$E_{pair} = 1/2\ a\left(1 - r_{ij}/r_{ij,cutoff}\right)^2, \quad r_{ij} < r_{ij,cutoff}$$

where a represents the pairwise interaction parameter, r_{ij} the distance between site i and site j, and $r_{ij,cutoff}$ the cutoff distance between these two sites. The default mixing rule is none (`mix -> none`).

7.12.11.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.11.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\cos(\theta) - \cos(\theta_0))^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.11.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 6
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \cos(n_i\phi - \delta_i)),$$

where k_i represents a set of torsion constants, δ_i a torsion offset, n_i the torsion pre-factor, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. The maximum

allowed value of n_i is 6. Note, that the functional form presented here differs from the original OPLS definition.

7.12.11.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.12 Gauss

7.12.12.1 Syntax

```
dpd             = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair	struct	Pair interaction descriptors

The Gaussian force field is a compounded force field, that finds its application in mesoscale simulations.[1] Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{gauss} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, improper, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/dpd/ for applications.

7.12.12.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    gauss       -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 20, l -> 1}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 5, theta -> 180}
        }
      },
      torsion   -> {
        active  -> true,
```

```
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 1,
        data    -> {
          {i0   -> a, i1 -> a, a -> 25.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> a, i1 -> b, a -> 25.2, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> a, i1 -> c, a -> 24.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> a, i1 -> d, a -> 26.4, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> b, i1 -> b, a -> 25.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> b, i1 -> c, a -> 21.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> b, i1 -> d, a -> 21.5, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> c, i1 -> c, a -> 25.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> c, i1 -> d, a -> 22.0, d -> 1, r0 -> 0, cutoff -> 1},
          {i0   -> d, i1 -> d, a -> 25.0, d -> 1, r0 -> 0, cutoff -> 1,
        }
      }
    }
  }
};
```

7.12.12.3 References

1. Reference.

7.12.12.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mode          -> option,
  cutoff        -> real,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      a         -> real,
      d         -> real,
      r0        -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`cutoff`	`real`	Sets global cut off for the `global` mode.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`a`	`real`	Energy scale constant.
`d`	`real`	Length scale constant.
`r0`	`real`	Length translation constant.
`cutoff`	`real`	Site size; the point where the potential equals zero.

The energetic functional form of pair contributions to the total potential is described by an exponential function,

$$E_{pair} = a_{ij} \exp\left[-1/4/d_{ij}^2 \ \left(r_{ij} - r_{0,ij}\right)^2\right], \quad r_{ij} < r_{ij,cutoff}$$

where a_{ij} represents the pairwise interaction parameter, d_{ij} a distance scaling factor, $r_{0,ij}$ a distance offset, r_{ij} the distance between sites i and j, and $r_{ij,cutoff}$ the cutoff distance between these two sites.

7.12.12.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.12.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\cos(\theta) - \cos(\theta_0))^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.12.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 6
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \cos(n_i\phi - \delta_i)),$$

where k_i represents a set of torsion constants, δ_i a torsion offset, n_i the torsion pre-factor, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. The maximum

allowed value of n_i is 6. Note, that the functional form presented here differs from the original OPLS definition.

7.12.12.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.13 GROMACS

7.12.13.1 Syntax

```
gromacs         = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The gromacs force field is a compounded force field based on the GROMACS force field.[1] Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{gromacs} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/nanoparticle/coarse/ for an application.

7.12.13.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    gromacs     -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
      torsion   -> {
```

```
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair14    -> {
        active  -> true
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.13.3 References

1. W. L. Jorgensen and J. Tirado-Rives, "*The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.13.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.13.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.13.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.13.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.13.8 Pair14

```
pair14          = {
  active        -> boolean,
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.

Representation of the 1-4 intra-molecular pair-wise contributions, which have the same functional form as the pair contributions in the following paragraph. The pre-factor, however is 0.5, as described in Jorgensen et al. [Jorgensen 1988].

7.12.13.9 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.14 MARTINI

7.12.14.1 Syntax

```
martini         = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The martini force field is a compounded force field based on the MARTINI force field.[1] Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{martini} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, improper, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/field/martini/ for an application.

7.12.14.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    martini     -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
```

```
      torsion   -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair14    -> {
        active  -> true
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.14.3 References

1. W. L. Jorgensen and J. Tirado-Rives, "*The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.14.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.14.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.14.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.14.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.14.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.15 Mie

7.12.15.1 Syntax

```
mie             = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair	struct	Pair interaction descriptors

The Mie force field is a compounded force field, that finds its application in mesoscale simulations.[1] Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{mie} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, improper, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/mie/ for applications.

7.12.15.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    mie         -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 20, l -> 1}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 5, theta -> 180}
        }
      },
      torsion   -> {
        active  -> true,
```

```
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 1,
        data    -> {
          {i0   -> a, i1 -> a, m -> 12, n -> 6,
           epsilon -> 1, sigma -> 1,
           cutoff -> 1},
          ...
        }
      }
    }
  }
};
```

7.12.15.3 References

1. Reference.

7.12.15.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      m         -> real,
      n         -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant.
`sigma`	`real`	Site size.

`m`	`real`	First exponent; m should be a positive number larger than 1.
`n`	`real`	Second exponent; n should smaller than m but larger than 1.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a m-n Lennard-Jones potential,

$$E_{pair} = \frac{m\ \epsilon}{m-n} \left(\frac{m}{n}\right)^{\frac{n}{m-n}} \left[\left(\frac{\sigma}{r_{ij}}\right)^m - \left(\frac{\sigma}{r_{ij}}\right)^n\right],$$

where r_{ij} represents the distance between site i and site j.

7.12.15.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.15.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\cos(\theta) - \cos(\theta_0))^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.15.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 6
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \cos(n_i\phi - \delta_i)),$$

where k_i represents a set of torsion constants, δ_i a torsion offset, n_i the torsion pre-factor, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. The maximum

allowed value of n_i is 6. Note, that the functional form presented here differs from the original OPLS definition.

7.12.15.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.16 OPLS

7.12.16.1 Syntax

```
opls            = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The OPLS force field is a compounded force field based on the OPLS force field.[1]. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{opls} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See `./examples/field/opls/` for applications.

7.12.16.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    opls        -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
      torsion   -> {
```

```
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair14    -> {
        active  -> true
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.16.3 References

1. W. L. Jorgensen and J. Tirado-Rives, "*The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.16.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is geometrical (`mix -> geometric`).

7.12.16.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.16.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.16.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} 1/2\ k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.16.8 Pair14

```
pair14          = {
  active        -> boolean,
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.

Representation of the 1-4 intra-molecular pair-wise contributions, which have the same functional form as the pair contributions in the following paragraph. The pre-factor, however is 0.5, as described in Jorgensen et al. [Jorgensen 1988].

7.12.16.9 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.17 SDK

7.12.17.1 Syntax

```
sdk             = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The SDK force field is a compounded force field based on the SDK force field.[1]. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{sdk} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See `./examples/field/sdk/` for applications.

7.12.17.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    sdk -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
      torsion   -> {
```

```
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair14    -> {
        active  -> true
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.17.3 References

1. Shinoda, Devane, Klein papers
2. W. L. Jorgensen and J. Tirado-Rives, "*The SDK Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.17.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.
`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.

`sigma`	`real`	Site size; the point where the potential equals zero.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.17.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.17.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.17.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} 1/2\ k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.17.8 Pair14

```
pair14          = {
  active        -> boolean,
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.

Representation of the 1-4 intra-molecular pair-wise contributions, which have the same functional form as the pair contributions in the following paragraph. The pre-factor, however is 0.5, as described in Jorgensen et al. [Jorgensen 1988].

7.12.17.9 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.18 Spline

7.12.18.1 Syntax

```
spline          = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
pair	struct	Pair interaction descriptors

The spline force field uses a set of splined functions for nonbonded and bonded interactions (see Section 7.9 [Splines], page 199). Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{spline} = E_{pair} + E_{bond} + E_{angle} + E_{torsion}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/field/spline/ for an application.

7.12.18.2 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      spline    -> struct
    },
    ...
  }
};
```

Directive	Parameters	Description
active	boolean	Interaction activator; either `true` or `false`.
n	integer	Number of data entries; can be ommited when `type = {merge -> true}`.
data	struct	Summary of interaction constants

i0, i1	id	Site id, referring to mass paragraph in types
spline	struct	Spline function structure.

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} =,$$

where

7.12.18.3 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      spline    -> struct
    },
    ...
  }
};
```

Directive	Parameters	Description
active	boolean	Interaction activator; either true or false.
n	integer	Number of data entries; can be ommited when type = {merge -> true}.
data	struct	Summary of interaction constants
i0, i1, i2	id	Site id, referring to mass paragraph in types
spline	struct	Spline function structure.

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} =,$$

where θ represents the bond angle.

7.12.18.4 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
```

```
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      spline    -> struct
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`
`spline`	`struct`	Spline function structure.

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} =,$$

where ϕ represents the bond torsion.

7.12.18.5 Pair

```
bond            = {
  active        -> boolean,
  nbonded       -> integer,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      spline    -> struct
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`spline`	`struct`	Spline function structure.

The energetic functional form of the pair contributions to the total potential is formed by a linear interpolation or a cubic spline through the provided data.

7.12.18.6 Example

```
variables       = {
};

simulation      = {
  types         -> {
    merge       -> true,
    boltzmann   -> {
      bond      -> {
        active  -> true,
        data    -> {
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        data    -> {
        }
      }
    }
  }
};
```

7.12.18.7 References

1. Spyriouni et al., Macromolecules 2007, 40, 3876

7.12.19 Standard

7.12.19.1 Syntax

```
standard        = {
  pair          -> struct,
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct
};
```

Directive	Parameters	Description
pair	struct	Pair interaction descriptors
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors

The standard force field is a compounded force field, that is OPLS-like in its appearance. Its functional form is a summation of nonbonded and bonded interactions given by

$$E_{standard} = E_{pair} + E_{bond} + E_{angle} + E_{torsion} + E_{improper}$$

for pair, bond, angle, torsion, and improper contributions respectively. The following paragraphs describe each contribution in detail. See ./examples/lj/ for an application. Note, that no standardized typing is provided for the standard force field.

7.12.19.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    standard    -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
      torsion   -> {
        active  -> true,
```

```
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09}
        }
      }
    }
  }
};
```

7.12.19.3 References

1. W. L. Jorgensen and J. Tirado-Rives, "*The OPLS Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.19.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential
`sigma`	`real`	Site size; the point where the potential equals zero

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.19.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2\ k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.19.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2\ k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.19.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 6
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \cos(n_i\phi - \delta_i)),$$

where k_i represents a set of torsion constants, δ_i a torsion offset, n_i the torsion pre-factor, and ϕ the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. The maximum

allowed value of n_i is 6. Note, that the functional form presented here differs from the original OPLS definition.

7.12.19.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.12.20 Table

7.12.20.1 Syntax

```
table           = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
pair	struct	Pair interaction descriptors

The tabular force field uses tabulated functions for nonbonded and bonded interactions. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{table} = E_{pair} + E_{bond} + E_{angle} + E_{torsion}$$

for bond, angle, torsion, and pair contributions respectively. The default interpolation order is set to `cubic` (see Section 7.9 [Splines], page 199). The following paragraphs describe each contribution in detail. See ./examples/field/table/ for an application.

7.12.20.2 Table Syntax

```
data            = {
  {
    x           -> real,
    energy      -> real
  },
  ...
};
```

Directive	Parameters	Description
x	real	Location; either interparticle distance for pair, bond distance for bond, bond angle of angle, or bond torsion for torsion interactions.
energy	real	Energy at the specified location.

Table entries are given in pairs, where the x coordinate represents the function variable and the y coordinate the resulting energy. The x interval between each entry must be equidistant. Non-equidistant entries result in poor interpolation.

7.12.20.3 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    boltzmann   -> {
      bond      -> {
        active  -> true,
        order   -> cubic,
        data    -> {
          {i0   -> a, i1 -> a, name -> "bond_aa.dat"},
          {i0   -> a, i1 -> b, name -> "bond_ab.dat"},
          {i0   -> b, i1 -> b, name -> "bond_bb.dat"}
        }
      },
      angle     -> {
        active  -> true,
        order   -> cubic,
        data    -> {
          {i0   -> a, i1 -> a, i2 -> a, name -> "angle_aaa.dat"},
          {i0   -> a, i1 -> a, i2 -> b, name -> "angle_aab.dat"},
          {i0   -> a, i1 -> b, i2 -> b, name -> "angle_abb.dat"},
          {i0   -> b, i1 -> b, i2 -> b, name -> "angle_bbb.dat"}
        }
      },
      pair      -> {
        active  -> true,
        order   -> cubic,
        nbonded -> 2,
        data    -> {
          {i0   -> a, i1 -> a, name -> "pair_aa.dat"},
          {i0   -> a, i1 -> b, name -> "pair_ab.dat"},
          {i0   -> b, i1 -> b, name -> "pair_bb.dat"}
        }
      }
    }
  }
};
```

7.12.20.4 Pair

```
bond            = {
  active        -> boolean,
  order         -> option,
  nbonded       -> integer,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      name      -> string
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`order`	`option`	Spline interpolation order; either `linear` or `cubic`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`name`	`string`	Table file name; for table format see [Table Syntax] below.

The energetic functional form of the pair contributions to the total potential is formed by a linear interpolation or a cubic spline through the provided data.

7.12.20.5 Bond

```
bond            = {
  active        -> boolean,
  order         -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      name      -> string
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`order`	`option`	Spline interpolation order; either `linear` or `cubic`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to `mass` paragraph in `types`
`name`	`string`	Table file name; for table format see [Table Syntax] below.

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = Interpolation(l),$$

where l represents the bond distance. Either a linear or a cubic spline through the provided data defines the interpolation function.

7.12.20.6 Angle

```
angle           = {
  active        -> boolean,
  order         -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      name      -> string
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`order`	`option`	Spline interpolation order; either `linear` or `cubic`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to `mass` paragraph in `types`
`name`	`string`	Table file name; for table format see [Table Syntax] below.

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = Interpolation(\theta),$$

where θ represents the bond angle. Either a linear or a cubic spline through the provided data defines the interpolation function.

7.12.20.7 Torsion

```
torsion         = {
  active        -> boolean,
  order         -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      name      -> string
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`order`	`option`	Spline interpolation order; either `linear` or `cubic`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to `mass` paragraph in `types`
`name`	`string`	Table file name; for table format see [Table Syntax] below.

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = Interpolation(\phi),$$

where ϕ represents the bond torsion. Either a linear or a cubic spline through the provided data defines the interpolation function.

7.12.21 TraPPE

7.12.21.1 Syntax

```
trappe          = {
  bond          -> struct,
  angle         -> struct,
  torsion       -> struct,
  improper      -> struct,
  pair14        -> struct,
  pair          -> struct
};
```

Directive	Parameters	Description
bond	struct	Bond interaction descriptors
angle	struct	Angle interaction decriptors
torsion	struct	Torsion interaction decriptors
improper	struct	Improper interaction decriptors
pair14	struct	1-4 intra-molecular pair interaction activator
pair	struct	Pair interaction descriptors

The TraPPE force field is a compounded force field based on the TraPPE force field.[1]. Its functional form is a summation of bonded and nonbonded interactions given by

$$E_{trappe} = E_{bond} + E_{angle} + E_{torsion} + E_{improper} + E_{pair}$$

for bond, angle, torsion, and pair contributions respectively. The following paragraphs describe each contribution in detail. See `./examples/field/trappe/` for applications.

7.12.21.2 Example

```
simulation      = {
  types         -> {
    merge       -> true,
    trappe      -> {
      bond      -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, k -> 100, l -> 2}
        }
      },
      angle     -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, k -> 100, theta -> 110}
        }
      },
      torsion   -> {
```

```
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, coefficients -> {
              {k -> 1.5, n -> 1, delta -> 0},
              {k -> 2.5, n -> 2, delta -> 0}
            }
          }
        }
      },
      improper  -> {
        active  -> true,
        data    -> {
          {i0   -> a, i1 -> b, i2 -> c, i3 -> d, k -> 100, psi -> 0}
        }
      },
      pair14    -> {
        active  -> true
      },
      pair      -> {
        active  -> true,
        nbonded -> 3,
        mode    -> global,
        cutoff  -> 10,
        mix     -> berthelot,
        data    -> {
          {i0   -> a, i1 -> a, sigma -> 4.0, epsilon -> 0.4},
          {i0   -> b, i1 -> b, sigma -> 3.7, epsilon -> 0.5},
          {i0   -> c, i1 -> c, sigma -> 4.1, epsilon -> 0.22},
          {i0   -> d, i1 -> d, sigma -> 2.4, epsilon -> 0.09},
        }
      }
    }
  }
};
```

7.12.21.3 References

1. Siepmann papers
2. W. L. Jorgensen and J. Tirado-Rives, "*The TraPPE Potential Functions for Proteins. Energy Minimizations for Crystals of Cyclic Peptides and Crambin*", *J. Am. Chem. Soc.* **1988**, *110*, 1657-1666.

7.12.21.4 Pair

```
pair            = {
  active        -> boolean,
  nbonded       -> integer,
  mix           -> option,
  shift         -> boolean,
  coulomb       -> option,
  cutoff        -> real,
  mode          -> option,
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      epsilon   -> real,
      sigma     -> real,
      cutoff    -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`nbonded`	`integer`	Number of bonded sites to exclude from nonbonded interactions; the given number is expected to be larger or equal to zero.
`mix`	`option`	Sets mixing rule; options are `none`, `berthelot`, `arithmetic`, `geometric`, and `sixth`.
`shift`	`boolean`	Sets shifting of potential at cutoff; options are `true` and `false`.
`coulomb`	`option`	Set type of coulomb treatment; options are `none`, `cut`, and `long`.
`cutoff`	`real`	Sets global cut off for `global` mode
`mode`	`option`	Sets cut off mode; options are `global` using the value of `cutoff` for all contributions, `individual` defining cut offs per contribution, and `repulsive` defining the well location for each separate contribution as the cut off distance.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph.

`epsilon`	`real`	Force interaction constant; identical to the well depth for a Lennard-Jones 6-12 potential.
`sigma`	`real`	Site size; the point where the potential equals zero.
`cutoff`	`real`	Cut off; distance at which the pair-wise contribution is set to zero; can be omitted.

The energetic functional form of the pair contributions to the total potential is described by a 6-12 Lennard-Jones potential,

$$E_{pair} = 4\epsilon \left(\frac{\sigma}{r_{ij}}\right)^6 \left[\left(\frac{\sigma}{r_{ij}}\right)^6 - 1\right],$$

where r_{ij} represents the distance between site i and site j. The default mixing rule is Berthelot (`mix -> berthelot`).

7.12.21.5 Bond

```
bond            = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      k         -> real,
      l         -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Bond spring constants
`l`	`real`	Equilibrium bond length

The energetic functional form of the bond contributions to the total potential is described by

$$E_{bond} = 1/2 k_{bond}(l - l_0)^2,$$

where k_{bond} represents a spring constant, l the bond length of bond $\{i_0, i_1\}$, and l_0 the equilibrium bond length.

7.12.21.6 Angle

```
angle           = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      k         -> real,
      theta     -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Angle spring constants
`theta`	`real`	Equilibrium bond angle

The energetic functional form of the angle contributions to the total potential is described by

$$E_{angle} = 1/2 k_{angle}(\theta - \theta_0)^2,$$

where k_{angle} represents a spring constant, θ the the angle between bonds $\{i_0, i_1\}$ and $\{i_1, i_2\}$, and θ_0 the equilibrium angle.

7.12.21.7 Torsion

```
torsion         = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      coefficients ->
      {
        {
          k     -> real,
          n     -> integer,
          delta -> real
        }
        ...
      }
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`n`	`integer`	Torsional angle prefactor; values range between 1 and 4
`delta`	`real`	Offset angle, mainly used to change the sign of the cosine function

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{torsion} = \sum_{i=1}^{m} k_i(1 + \text{sign}(n_i)\cos(n_i\phi - \delta_i)),$$

where ϕ represents the bond torsion constructed by bonds $\{i_0, i_1\}$, $\{i_1, i_2\}$, and $\{i_2, i_3\}$. Function $sign(n_i)$ is +1 for odd and -1 for even values of n_i.

7.12.21.8 Improper

```
improper        = {
  active        -> boolean
  n             -> integer,
  data          ->
  {
    {
      i0        -> id,
      i1        -> id,
      i2        -> id,
      i3        -> id,
      k         -> real,
      psi       -> real
    },
    ...
  }
};
```

Directive	Parameters	Description
`active`	`boolean`	Interaction activator; either `true` or `false`.
`n`	`integer`	Number of data entries; can be ommited when `type = {merge -> true}`.
`data`	`struct`	Summary of interaction constants
`i0, i1, i2, i3`	`id`	Site id, referring to types as defined by the `mass` paragraph in `types`
`k`	`real`	Force constant
`psi`	`real`	Improper equilibrium angle

The energetic functional form of the torsion contributions to the total potential is described by

$$E_{improper} = k_{improper}(\psi - \psi_0)^2,$$

where ψ represents the improper angle, which is given by the average of torsion angles formed by planes out of permutations of any of the following two bonds: $\{i_1, i_0\}$, $\{i_2, i_0\}$, and $\{i_3, i_0\}$. Note, that site i_0 is the central site, as opposed to some representations, where i_1 is the central site.

7.13 Units

7.13.1 Syntax

```
units           = {
  type          -> constant,
  mass          -> real,
  length        -> real,
  angstrom      -> real,
  angle         -> real,
  energy        -> real,
  kb            -> real,
  nav           -> real,
  charge        -> real,
  permittivity  -> real,
  seed          -> real,
  reduced       -> boolean
};
```

Directive	Parameters	Description
`type`	`constant`	Defines the type of units used (needed upon first initialization); options are `reduced` or `si`.
`mass`	`real`	Defines the global mass scaling constant.
`length`	`real`	Defines the global length scaling constant.
`angstrom`	`real`	Defines the length scaling for ported formats (e.g. PDB or LAMMPS).
`angle`	`real`	Defines the global angle scaling constant.
`energy`	`real`	Defines the global energetic scaling constant.
`kb`	`real`	Defines Boltzmann's constant with respect to the chosen scaling.
`nav`	`real`	Defines Avogadro's constant with respect to the chosen scaling.
`charge`	`real`	Defines the global scaling of a unit charge.
`permittivity`	`real`	Defines the global permittivity scaling.
`seed`	`real`	Defines the random seed.
`reduced`	`boolean`	Reflects the output state (normally `false`); options are `true` or `false`.

7.13.2 Usage

This variable style describes units. Units are used to internally scale all variables with respect to their units. The units can also be used to define an appropriate relation between force field energies, angles, bead sizes, etc. Upon initialization, a suitable scaling will be deduced from force field or internal natural constant definitions when the contributor is zero (see Section 7.12 [Types], page 204).

7.13.3 Default

Unless otherwise stated, the default is given by

```
units           = {
  type          -> si
};
```

By default, all other contributors are set to zero and deduced upon initialization.

7.14 Vector

7.14.1 Syntax

```
vector          = {x -> real, y -> real, z -> real};
```

Directive	Parameters	Description
x, y, z	real	Sets vector components.

7.14.2 Usage

This variable style describes a vector. Vectors always consist of three components, which can be given in real numbers. Interpretation can either be `real` or `integer`, dependent on the command in which `vector` occurs.

7.14.3 Default

Unless otherwise stated, the default is given by

```
vector          = {x -> 0, y -> 0, z -> 0};
```

7.15 Voigt

7.15.1 Syntax

```
voigt           = {xx -> real, yy -> real, zz -> real,
                   zy -> real, zx -> real, yx -> real};
```

Directive	Parameters	Description
`xx, yy, zz, zy, zx, yx`	`real`	Sets voigt components.

7.15.2 Usage

This variable style describes a voigt. Voigts always consist of six components, which can be given in real numbers. Interpretation can either be `real` or `integer`, dependent on the command in which `voigt` occurs.

7.15.3 Default

Unless otherwise stated, the default is given by

```
voigt           = {xx -> 0, yy -> 0, zz -> 0, zy -> 0, zx -> 0, yx -> 0};
```

Index

P

R

S

T

U

V

X