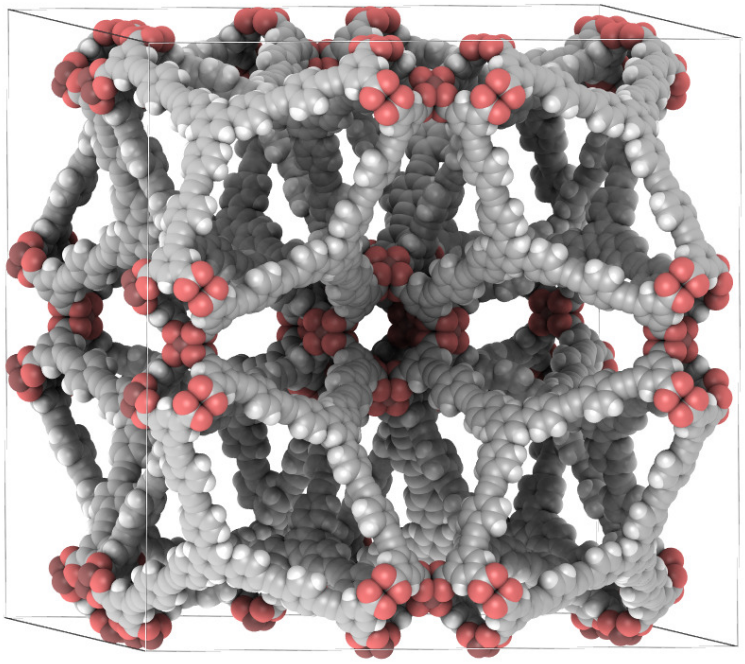
RASPA 3.0: Molecular Software Package for Adsorption and Diffusion in (Flexible) Nanoporous Materials



Youri Ran
email: y.a.ran@uva.nl
University of Amsterdam

Shrinjay Sharma
email: S.Sharma-6@tudelft.nl
Delft University of Technology

Salvador Rodríguez Gómez
email: salrodgom@upo.es
Universidad Pablo de Olavide

Zhao Li
email: zhaoli2023@u.northwestern.edu
Northwestern University

Sofia Calero
email: s.calero@tue.nl
Eindhoven University of Technology

Thijs J.H. Vlugt
email: t.j.h.vlugt@tudelft.nl
Delft University of Technology

Randall Q. Snurr
email: snurr@northwestern.edu
Northwestern University

David Dubbeldam
email: d.dubbeldam@uva.nl
University of Amsterdam

August 7, 2024

Contents

1 Introduction

1.1 Design philosophy

RASPA3 is a molecular simulation code for computing adsorption and diffusion in nanoporous materials, and thermodynamic and transport properties of fluids. It implements force field based classical Monte Carlo/Molecular Dynamics in various ensembles. RASPA3 can be used for the simulation of molecules in gases, fluids, zeolites, aluminosilicates, metal-organic frameworks, and carbon nanotubes. One of the main applications of RASPA3 is to compute adsorption isotherms and to understand the atomic-level mechanism of adsorption and separations.

RASPA3 is redesigned and rewritten from the ground up in C++23, based on the following ideas:

- Composition and value semantics
  Major improvements in efficiency result from the implementation of C++ code based on composition and value semantics (in contrast to inheritance and reference semantics). Composition is a structural approach in which complex objects are built from simple building blocks. Value semantics avoid sharing mutable state and upholds the independence of values to support local reasoning. References are only used implicitly, at function boundaries, and are never stored. This style of coding has many advantages. Avoiding complex inheritance hierarchies and reducing the need for virtual functions lead to major code simplifications. It encourages code clarity and local reasoning. With composition, components are directly included in an object, removing the necessity for pointers, manual memory allocations, and indirections. This leads to straightforward memory patterns and access safety. Lifetimes of objects are coupled with composition, eliminating the need for explicit memory management. These lead to better compiler optimization and therefore performance of the code.

- Correctness and accuracy
  For all the techniques and algorithms available in RASPA3 we have implemented the 'best' ones available in literature. For example, RASPA3 uses Configurational-Bias Monte-Carlo, it uses the Ewald summation for electrostatics, molecular dynamics is based on 'symplectic' integrators, all Monte-Carlo moves obey detailed balance etc. Unit tests test the smallest functional units of code and prevent developers breaking existing code when refactoring or adding new code. The unit tests in RASPA3 are arranged hierarchically. Atoms are placed at several distances and the Lennard-Jones potentials are tested and compared with the analytically computed energy. Then molecules are placed within a zeolite framework at predetermined positions and compared to the energies computed with RASPA2. The various energy routines used in biased-sampling are tested by comparing to the general routines.

The various routines for the gradients are tested by comparing the computed gradients to the values computed by finite difference schemes based on the energy. Likewise, the strain-derivative tensor (related to the pressure) is tested by comparing to the values computed by finite difference schemes based on the energy of a strained cell.

- Input made easy
  The input of RASPA3 has been changed to JSON format. JSON stands for JavaScript Object Notation. It is a lightweight data-interchange format in plain text written in JavaScript object notation. It is language independent. The requirements for the input files is kept as minimal as possible. Only for more advanced options extra commands in the input file are needed. Also the format of the input is straightforward. Default settings are usually the best ones. Fugacity coefficients and excess adsorption are automatically computed.
- Output made easy
  Similarly, the JSON format is now also used for the output. Where previously all initial data (e.g. hardware info, unit conversion factors) and statistics (e.g. CPU timings, MC move statistics) were written to a text file, these are now written as a nested dictionary to a JSON file.
- Integrated simulation environment
  The RASPA3 C++ simulation engine is made available to Python via a Pybind11 interface. For use in Python, RASPA3 is built as a shared library, allowing its functions to be used by Python users and enabling seamless interactions with the simulation routines. Through the API, the library allows for invocation of RASPA3's simulation routines from Python scripts, calling the same simulation routines as via the JSON input. This execution directly from Python enables the ability to prototype simulation settings and incorporate RASPA3 into existing workflows. Extension and modification of the code is relatively straightforward.

1.2 Units and conventions

- The standard units in RASPA from which all other units are derived are:

quantity	symbol	unit	value
length	l	Ångstrom	10^{-10} m
temperature	T	Kelvin	K
mass	m	atomic mass	$1.6605402 \times 10^{-27}$ kg
time	t	pico seconds	10^{-12} s
charge	q	atomic charge	$1.60217733 \times 10^{-19}$ C/particle

Some examples of derived units:

quantity	symbol	units	conversion value
energy	U	$J = \text{mass} \times \text{length}^2/\text{time}^2$	1.66054×10^{-23} (=10 J/mol)
pressure	p	$\text{Pa} = \text{mass}/(\text{length} \times \text{time}^2)$	1.66054×10^{7}
diffusion constant	D	$D = \text{length}^2/\text{time}$	1×10^{-8}
force	f	$f = \text{length}/\text{time}^2$	1.66054×10^{-13}
...	...	...	...

A pressure input of 10 Pascal in the input file, is converted to 'internal units' by dividing by 1.66054×10^7. In the output any internal pressure is printed, multiplied by 1.66054×10^7. It is not necessary to convert units besides input and output, with a few exceptions. One of them is the Coulombic conversion factor

$$\frac{q_i q_j}{4\pi\epsilon_0} = \frac{\text{charge}^2}{4\pi \times \text{electric constant} \times \text{length} \times \text{energy}} = 138935.4834964017 \tag{1.1}$$

with the electric constant as $8.8541878176 \times 10^{-12}$ in units of $C^2/(N.m^2)$. This factor is needed to convert the electrostatic energy to the internal units at every evaluation.

The Boltzmann's constant k_B is

$$k_B = \text{Boltzmann constant/energy} = 0.8314464919 \tag{1.2}$$

with the Boltzmann constant as $1.380650324 \times 10^{-23}$ in units of J/K, and $k_B = 0.8314464919$ in internal units.

- Numbering is based on the C-convention, i.e. starting from zero.
- Files in the current directory always have preference.
  Sometimes one would like to try various parameters for force field fitting for example. In order to avoid making a lot of directories for each force field it is more convenient to have the 'pseudo_atoms.def', 'force_field_mixing_rule.def' and 'force_field.def' files in the *current* directory.

1.3 Compiling and installing RASPA

1.3.1 Requirements

CMake 3.28 and later support C++ modules. C++20 named modules are now supported by the Ninja Generator 1.11 or newer in combination with the LLVM/Clang 16.0 and newer, MSVC toolset 14.34 and newer, or GCC 14 and newer. We recommend LLVM/Clang 18 or higher for compiling RASPA3. This version has support for `std::format`, `std::print`, and `std::jthread`. The output-files of RASPA3 are in UTF-8 encoding and contain unicode characters (e.g. for Å). RASPA3 depends on

- C++23 compliant compiler
- Cmake 3.28
- Ninja Generator 1.11
- Openmp
- hdf5
- lapack and blas (64-bit integers)
- pybind11

1.3.2 RASPA from 'git'

Working with 'git' and a remote repository means that you will have to distinguish between two locations of the code:

1. The repository (visible to everyone)
2. your local copy (only visible to you)

To check-out the code for the first time do:

```
git clone https://github.com/iraspa/RASPA3
```

After that, you can update the code by using

```
git pull
```

1.3.3 Installing RASPA

Download one of the precompiled packages from

```
https://github.com/iRASPA/RASPA3/releases
```

In Figure 1 one can find the list of packages. These include packages for macOS (both intel and apple silicon), and many linux distributions

1.3.4 Compiling RASPA

Using `cmake --list-presets` you can see the list of CMake presets. These include

- ”macos-intel”
- ”macos-intel-debug”
- ”macos-apple-silicon”
- ”macos-apple-silicon-debug”
- ”linux”
- ”linux-opensuse-leap-15.2”
- ”linux-opensuse-leap-15.3”
- ”linux-opensuse-leap-15.4”
- ”linux-opensuse-leap-15.5”
- ”linux-opensuse-tumbleweed”
- ”linux-archlinux”
- ”linux-redhat-6”
- ”linux-redhat-7”
- ”linux-redhat-8”
- ”linux-redhat-9”
- ”linux-debian-12”
- ”linux-debian-11”
- ”linux-debian-10”
- ”linux-ubuntu-24”
- ”linux-ubuntu-22”
- ”linux-ubuntu-20”
- ”linux-fedora-35”
- ”linux-fedora-36”

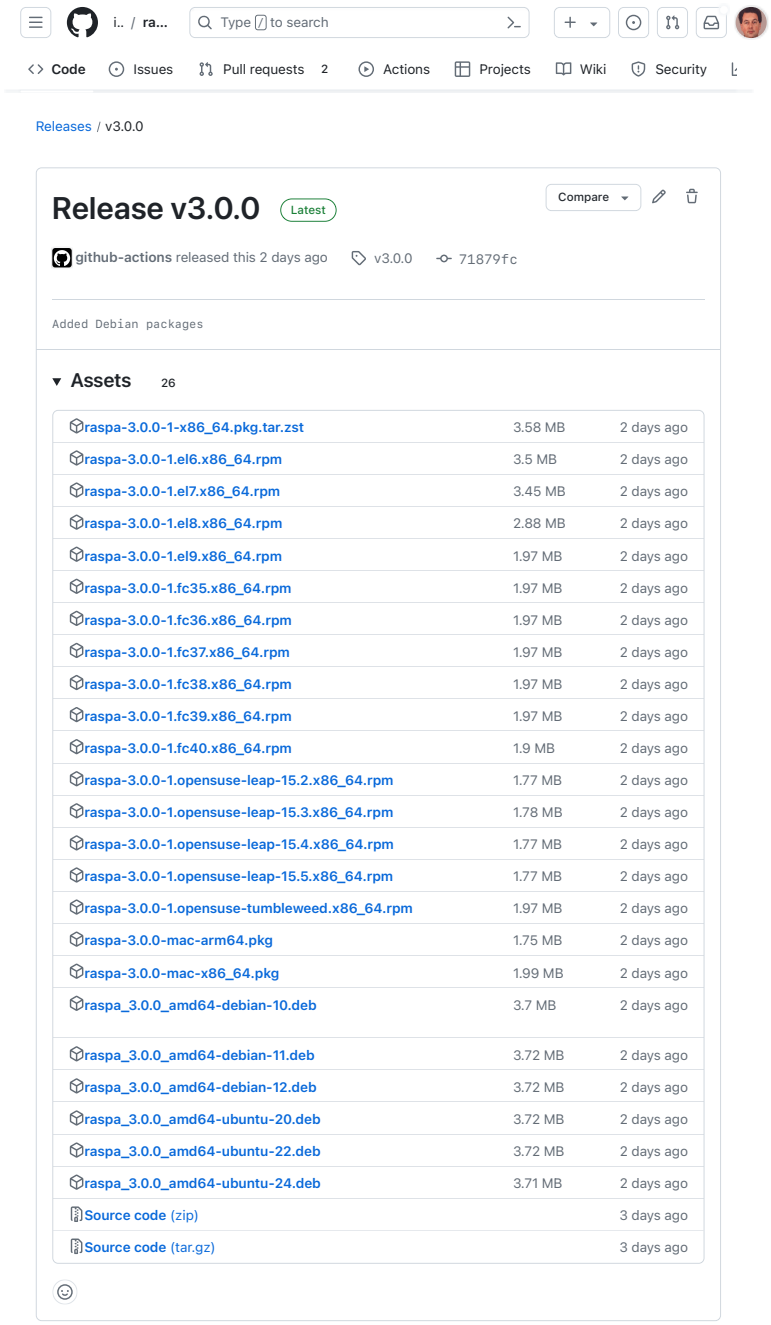


Figure 1: *List of available binary packages: Linux distributions based on Red Hat and OpenSUSE (rpm-packages), Debian (deb-packages), and Arch Linux (tar.zst packages), and installers for intel- and apple-silicon macs.*

- ″linux-fedora-37″
- ″linux-fedora-38″
- ″linux-fedora-39″
- ″linux-fedora-40″

Installation on macOS is then accomplished for example by

```
cmake -B build --preset macos-intel
ninja -C build install
```

or on linux

```
cmake -B build --preset linux-ubuntu-22
ninja -C build install
```

afterwhich the unit tests can be run

```
ctest --test-dir build/tests --verbose
```

Packages can be created with

```
ninja -C build package
```

Doxygen code documentation can be created with

```
ninja -C build documentation
```

Installing required packages on Ubuntu-24 or higher

```
apt-get install -y git ca-certificates cmake ninja-build
apt-get install -y llvm lld clang clang-tools clang-tidy
apt-get install -y libc++-dev libc++abi-dev libomp-dev libclang-rt-dev
apt-get install -y python3 pybind11-dev python3-pybind11 python3-dev
apt-get install -y liblapack64-dev libblas64-dev
```

Installing required packages on Fedora-40 or higher

```
dnf install -y wget git rpm-build
dnf install -y llvm lld cmake clang clang-tools-extra ninja-build
dnf install -y libcxx libcxxabi libcxx-devel libcxxabi-devel
dnf install -y libomp-devel libcxx-static libcxxabi-static
dnf install -y lapack-devel lapack64 blas64
dnf install -y python3 python3-devel python3-pybind11
dnf install -y pybind11-devel
```

Installing required packages on macOS using Homebrew

```
dnf install -y wget git rpm-build
dnf install -y llvm lld cmake clang clang-tools-extra ninja-build
dnf install -y libcxx libcxxabi libcxx-devel libcxxabi-devel
dnf install -y libomp-devel libcxx-static libcxxabi-static
dnf install -y lapack-devel lapack64 blas64
dnf install -y python3 python3-devel python3-pybind11
dnf install -y pybind11-devel
```

1.3.5 Running RASPA

Running RASPA is based on two files:

- A 'run' file to execute the program
  an example file is:

  ```
  #! /bin/sh -f
  export RASPA_DIR=/usr/
  ${RASPA_DIR}/bin/raspa3
  ```

  This type of file is know as a 'shell script'. RASPA needs the variable 'RASPA_DIR' to be set in order to look up the molecules, frameworks, etc. The scripts sets the variable and runs RASPA. RASPA can then be run from any directory you would like.

- An 'input'-file describing the type of simulation and the parameters
  In the same directory as the 'run'-file, there needs to be a file called `simulation.json`. An example file is:

  ```
  {
    "SimulationType" : "MonteCarlo",
    "NumberOfCycles" : 100000,
    "NumberOfInitializationCycles" : 1000,
    "NumberOfEquilibrationCycles" : 10000,
    "PrintEvery" : 1000,

    "Systems" :
    [
      {
        "Type" : "Box",
        "BoxLengths" : [30.0, 30.0, 30.0],
        "ExternalTemperature" : 300.0,
        "ChargeMethod" : "None",
  ```

```
      "OutputPDBMovie" : true,
      "SampleMovieEvery" : 10
    }
  ],

  "Components" :
  [
    {
      "Name" : "methane",
      "MoleculeDefinition" : "ExampleDefinitions",
      "TranslationProbability" : 1.0,
      "CreateNumberOfMolecules" : 100
    }
  ]
}
```

This tells RASPA to run a Monte-Carlo simulation of 100 methane molecules in a $30 \times 30 \times 30$ Å cubic box (with 90° angles) at 300 Kelvin. It will start with 1000 cycles to initialize the system, 10000 cycles to equilibrate the system, and will use 100000 cycle to obtain thermodynamic properties of interest. Every 1000 cycles a status-report is printed to the output. The Monte-Carlo program will use only the 'translation move' where a particle is given a random translation and the move is accepted or rejected based on the energy difference.

In order to run it on a cluster using a queuing system one needs an additional file 'bsub.job' (arbitrary name)

- 'gridengine'

```
#!/bin/bash
# Serial sample script for Grid Engine
# Replace items enclosed by {}
#$ -S /bin/bash
#$ -N Test
#$ -V
#$ -cwd
echo $PBS_JOBID > jobid
exprot RASPA_DIR=/usr
$RASPA_DIR/bin/raspa3
```

The job can be submitted using 'qsub bsub.job'.

- 'torque'

```
#!/bin/bash
#PBS -N Test
#PBS -o pbs.out
#PBS -e pbs.err
#PBS -r n
#PBS -V
#PBS -mba
cd $PBS_O_WORKDIR
echo $PBS_JOBID > jobid
```

```
export RASPA_DIR=/usr
${RASPA_DIR}/bin/raspa3
```

The job can be submitted using 'qsub bsub.job'.

- 'slurm'

```
#!/bin/bash
#SBATCH -N 1
#SBATCH --job-name=Test
#SBATCH --export=ALL
echo $SLURM_JOBID > jobid
valhost=$SLURM_JOB_NODELIST
echo $valhost > hostname
module load slurm
export RASPA_DIR=/usr
${RASPA_DIR}/bin/raspa3
```

The job can be submitted using 'sbatch bsub.job'.

1.4 Output from RASPA

RASPA generates output from the simulation. Some data is just information on the status, while other data are written because you specifically asked the program to compute it for you. The output is written to be used with other programs like:

- gnuplot
- iRASPA
- VMD

1.5 Citing RASPA

If you are using RASPA and would like to cite it in your journal articles or book-chapters, then for RASPA:

Y.A. Ran, S. Sharma, S.R.G. Balestra, Z. Li, S. Calero, T.J.H Vlugt, R.Q. Snurr, and D. Dubbeldam RASPA3: A Monte Carlo Code for Computing Adsorption and Diffusion in Nanoporous Materials and Thermodynamics Properties of Fluids *J. Chem. Phys.*, 2024.

D. Dubbeldam, S. Calero, D.E. Ellis, and R.Q. Snurr, RASPA: Molecular Simulation Software for Adsorption and Diffusion in Flexible Nanoporous Materials, *Mol. Simulat.*, http://dx.doi.org/10.1080/08927022.2015.1010082, 2015.

For the inner workings of Monte Carlo codes:

D. Dubbeldam, A. Torres-Knoop, and K.S. Walton, On the Inner Workings of Monte Carlo Codes, http://dx.doi.org/10.1080/08927022.2013.819102 *Mol. Simulat.*, 39(14-15), 1253-1292, 2013.

For the description of Molecular Dynamics and diffusion:

D. Dubbeldam and R.Q. Snurr, Recent Developments in the Molecular Modeling of Diffusion in Nanoporous Materials, http://dx.doi.org/10.1080/08927020601156418, *Mol. Simulat.*, 33(4-5), 305-325, 2007.

For the description of the implementation of force fields:

D. Dubbeldam, K.S. Walton, T.J.H. Vlugt, and S. Calero, Design, Parameterization, and Implementation of Atomic Force Fields for Adsorption in Nanoporous Materials, https://doi.org/10.1002/adts.201900135, *Adv. Theory Simulat.*, 2(11), 1900135, 2019.

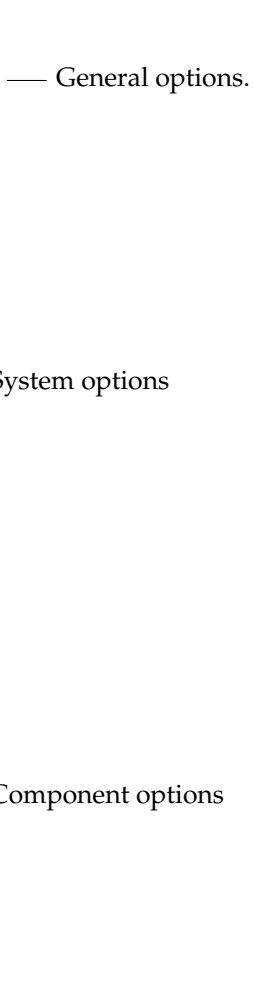
2 Simulation input options

2.1 Input sections

```
{
  "SimulationType" : "MolecularDynamics",
  "NumberOfCycles" : 100000,
  "NumberOfInitializationCycles" : 1000,
  "NumberOfEquilibrationCycles" : 10000,
  "PrintEvery" : 1000,

  "Systems" : [
    {
      "Type" : "Framework",
      "Name" : "Cu-BTC",
      "NumberOfUnitCells" : [1, 1, 1],
      "ChargeMethod" : "Ewald",
      "ExternalTemperature" : 323.0,
      "ExternalPressure" : 1.0e4,
      "OutputPDBMovie" : false,
      "SampleMovieEvery" : 10
    }
  ],

  "Components" : [
    {
      "Name" : "CO2",
      "FugacityCoefficient" : 1.0,
      "TranslationProbability" : 0.5,
      "RotationProbability" : 0.5,
      "ReinsertionProbability" : 0.5,
      "SwapProbability" : 0.0,
      "WidomProbability" : 0.0,
      "CreateNumberOfMolecules" : 20
    }
  ]
}
```



Listing 1: *Structure and input sections of the* `simulation.json` *file.*

2.2 General options

2.2.1 Simulation types

- `"SimulationType" : "MonteCarlo"`
  Starts the Monte Carlo part of RASPA. The particular ensemble is not specified but implicitly deduced from the specified Monte Carlo moves. Note that a MD-move can be used for hybrid MC/MD.

- `"SimulationType" : "MolecularDynamics"`
  Starts the Molecular Dynamics part of RASPA. The ensemble must be explicitly specified.

2.2.2 Simulation duration

- `"NumberOfCycles" : integer`
  The number of cycles for the production run. For Monte Carlo a cycle consists of N steps, where N is

the amount of molecules with a minimum of 20 steps. This means that on average during each cycle on each molecule a Monte Carlo move has been attempted (either successful or unsuccessful). For MD the number of cycles is simply the amount of integration steps.

- `"NumberOfInitializationCycles": integer`
  The number of cycles used to initialize the system using Monte Carlo. This can be used for both Monte Carlo as well as Molecular Dynamics to quickly equilibrate the positions of the atoms in the system.

- `"NumberOfEquilibrationCycles" : integer`
  For Molecular Dynamics it is the number of MD steps to equilibrate the velocities in the systems. After this equilibration the production run is started. For Monte Carlo, in particular CFCMC, the equilibration-phase is used to measure the biasing factors.

2.2.3 Restart and crash-recovery

- `"RestartFile" : boolean`
  Reads the positions, velocities, and force from the directory 'RestartInitial'. Any creation of molecules in the 'simulation.input' file will be in addition and after this first read from file. This is useful to load initial positions of cations for example, and after that create adsorbates. The restart file is written at 'PrintEvery' intervals.

- `"ContinueAfterCrash" : boolean`
  Write a binary file containing the complete status of the program. The file name is 'binary_restart.dat' and is located in the directory 'CrashRestart'. With this option to `true` the presence of this file will result in continuation from the point where the program was at the moment of outputting this file. The file can be quite big (several hundreds of megabytes) and will be outputted every 'WriteBinaryRestartFileEvery' cycles.

- `"WriteBinaryRestartFileEvery" : integer`
  The output frequency (i.e. every [integer] cycles) of writing the crash-recovery file.

2.2.4 Printing options

- `"PrintEvery" : integer`
  Prints the loadings (when a framework is present) and energies every [integer] cycles. For MD information like energy conservation and stress are printed.

2.3 System options

2.3.1 Operating conditions and thermostat/barostat-parameters

- `"ExternalTemperature" : floating-point-number`
  The external temperature in Kelvin for the system. Default: 298 K.

- `"ExternalPressure" : floating-point-number`
  The external pressure in Pascal for the system. Default: 0 Pa.

- `"ThermostatChainLength" : integer`
  The length of the chain to thermostat the system. Default: 5.

- `"NumberOfYoshidaSuzukiSteps" : integer`
  The number of Yoshida/Suzuki multiple timesteps. Default: 5.

- `"TimeScaleParameterThermostat" : floating-point-number`
  The time scale on which the system thermostat evolves. Default: 0.15 ps.

2.3.2 Box/Framework options

- `"Type" : string`
  Sets the system type. The type string can be:
  - `"Box"`
    Sets the system to a simulation cell where the lengths and angles of the cell can be explicitely be specified.
  - `"Framework"`
    Set the system to type 'Framework'. The cell lengths and cell angles follows from the specified framework file.
- `"BoxLengths" : [floating-point-number, floating-point-number, floating-point-number]`
  The cell dimensions of rectangular box of system in Angstroms.
- `"BoxAngles" : [floating-point-number, floating-point-number, floating-point-number]`
  The cell angles of rectangular box of system in Degrees.
- `"Name" : string`
  Loads the framework with name `string.cif`.
- `"NumberOfUnitCells" : [integer, integer, integer]`
  The number of unit cells in `x`, `y`, and `z` direction for the system. The super-cell will contain the unit cells, and periodic boundary conditions will be applied on the super-cell level (*not* on a unit cell level).
- `"HeliumVoidFraction" : floating-point-number`
  Sets the void fraction as measured by probing the structure with helium a room temperature. This quantity has to be obtained from a separate simulation and is essential to compute the *excess*-adsorption during the simulation.

2.3.3 Force field definitions

- `"ForceField" : string`
  Reads in the force field file `string.json`, Note that if this file is in the working directory then this will be read and used instead of:

  `${RASPA_DIR}/simulations/share/raspa3/forcefield/string/force_field.json`
- `"CutOffVDW" : floating-point-number`
  The cutoff of the Van der Waals potentials. Interactions longer then this distance are omitted from the energy and force computations. The potential can either be shifted to zero at the cutoff, or interactions can just neglected after the cut off, or the remainder of the potential energy can be approximated using tail corrections. This is specified in the force field files and can be specified globally or for each interaction individually.
- `"CutOffCoulombic" : floating-point-number`
  The cutoff of the charge-charge potential. The potential is truncated at the cutoff. No tail-corrections are (or can be) applied. The only way to include the long-range part is to use 'ChargeMethod Ewald'. The parameter is also used in combination with the Ewald precision to compute the number of wave vectors and Ewald parameter α. For the Ewald summation using rather large unit cells, a charge-charge cutoff of about half the smallest box-length would be advisable in order to avoid the use of an excessive amount of wave-vectors in Fourier space. For non-Ewald methods the cutoff should be as large as possible (greater than about 30 Å).

- `"ChargeMethod" : string` Sets the method to compute charges. The string can be:
  - `"None"`
    Skips the entire charge calculation and should only be used when all adsorbates do not contain any charges.
  - `"Ewald"`
    Switches on the Ewald summation for the charge calculation.
- `"UseChargesFromCIFFile" : boolean`

2.3.4 System MC-moves

- `"VolumeChangeProbability" : floating-point-number`
  The probability per cycle to attempt a volume-change. Rigid molecules are scaled by center-of-mass, while flexible molecules and the framework is atomically scaled.
- `"GibbVolumeChangeProbability" : floating-point-number`
  The probability per cycle to attempt a Gibbs volume-change MC move during a Gibbs ensemble simulation. The total volume of the two boxes (usually one for the gas phase, one for the liquid phase) remains constant, but the individual volume of the boxes are changed. The volumes are changed by a random change in $\ln(V_I/V_{II})$.

2.3.5 Molecular dynamics parameters

- `"TimeStep" : floating-point-number`
  The time step in picoseconds for MD integration. Default value: 0.0005 ps (0.5 fs).
- `"Ensemble" : string`
  Sets the ensemble. The ensemble string can be:
  - `"NVE"`
    The micro canonical ensemble, the number of particle N, the volume V, and the energy E are constant.
  - `"NVT"`
    The canonical ensemble, the number of particle N, the volume V, and the average temperature $\langle T \rangle$ are constant. Instantaneous values for the temperature are fluctuating.

2.3.6 Options to measure properties

Output pdb-movies

```
"OutputPDBMovie" : boolean
```

Sets whether or not to output simulation snapshots to pdb movies.
Output is written to the directory `movies`.

- `"SampleMovieEvery" : integer`
  Sample the movie every [integer] cycles. Default: 1.

Histogram of the energy

```
"ComputeEnergyHistogram" : boolean
```

Sets whether or not to compute a histogram of the energy for the current system. For example, during adsorption it keeps track of the total energy, the VDW energy, the Coulombic energy, and the polarization energy.
Output is written to the directory `energy_histogram`.

- `"SampleEnergyHistogramEvery" : integer`
  Sample the energy histogram of the system every [integer] cycles. Default: 1.
- `"WriteEnergyHistogramEvery" : integer`
  Writes the energy histogram of the system every [integer] cycles. Default: 5000.
- `"NumberOfBinsEnergyHistogram" : integer`
  Sets the number of elements of the histogram. Default: 128.
- `"LowerLimitEnergyHistogram" : floating-point-number`
  The lower limit of the histogram. Default: -5000.
- `"UpperLimitEnergyHistogram" : floating-point-number`
  The upper limit of the histogram. Default: 1000.

Histogram of the number of molecules

```
"ComputeNumberOfMoleculesHistogram" : boolean
```

Sets whether or not to compute the histograms of the number of molecules for the current system. In open ensembles the number of molecules fluctuates.
Output is written to the directory `number_of_molecules_histogram`.

- `"SampleNumberOfMoleculesHistogramEvery" : integer`
  Sample the histogram every [integer] cycles. Default: 1.
- `"WriteNumberOfMoleculesHistogramEvery" : integer`
  Output the histogram every [integer] cycles. Default: 5000.
- `"LowerLimitNumberOfMoleculesHistogram" : floating-point-number`
  The lower limit of the histograms. Default: 0.
- `"UpperLimitNumberOfMoleculesHistogram" : floating-point-number`
  The upper limit of the histograms. Default: 200.

Radial Distribution Function (RDF) force-based

```
"ComputeRDF" : boolean
```

Sets whether or not to compute the radial distribution function (RDF).
Output is written to the directory `rdf`.

- `"SampleRDFEvery" : integer`
  Sample the rdf every [integer] cycles. Default: 10.

- `"WriteRDFEvery" : integer`
  Output the rdf every [integer] cycles. Default: 5000.

- `"NumberOfBinsRDF" : integer`
  Sets the number of elements of the rdf. Default: 128.

- `"UpperLimitRDF" : floating-point-number`
  The upper limit of the rdf. Default: 15.0.

Radial Distribution Function (RDF) conventional

```
"ComputeConventionalRDF" : boolean
```

Sets whether or not to compute the radial distribution function (RDF).
Output is written to the directory `conventional_rdf`.

- `"SampleConventionalRDFEvery" : integer`
  Sample the rdf every [integer] cycles. Default: 10.

- `"WriteConventionalRDFEvery" : integer`
  Output the rdf every [integer] cycles. Default: 5000.

- `"NumberOfBinsConventionalRDF" : integer`
  Sets the number of elements of the rdf. Default: 128.

- `"UpperLimitConventionalRDF" : floating-point-number`
  The upper limit of the rdf. Default: 15.0.

Mean-Squared Displacement (MSD) order-N

```
"ComputeMSD" : boolean
```

Sets whether or not to compute the mean-squared displacement (MSD).
Output is written to the directory `msd`.

- `"SampleMSDEvery" : integer`
  Sample the msd every [integer] cycles. Default: 10.

- `"WriteMSDEvery" : integer`
  Output the msd every [integer] cycles. Default: 5000.

- `"NumberOfBlockElementsMSD" : integer`
  The number of elements per block of the msd. Default: 25.0.

Density grids

```
"ComputeDensityGrid" : boolean
```

Sets whether or not to compute the density grids.
Output is written to the directory `density_grids`.

- `"SampleDensityGridEvery" : integer`
  Sample the density grids every [integer] cycles. Default: 10.

- `"WriteDensityGridEvery" : integer`
  Output the density grids every [integer] cycles. Default: 5000.

- `"DensityGridSize" : [integer, integer, integer]`
  Sets the size of the density grids. Default: [128, 128, 128].

- `"DensityGridPseudoAtomsList" : [string]`
  The list of pseudo-atoms for which to compute the density grids.

2.4 Component options

2.4.1 Component properties

- `"Name" : string`
  The descriptive name of the component.

- `"MolFraction" : floating-point-number`
  The mol fraction of this component in the mixture. The values can be specified relative to other components, as the fractions are normalized afterwards. The partial pressures for each component are computed from the total pressure and the mol fraction per component.

- `"FugacityCoefficient" : floating-point-number`
  The fugacity coefficient for the current component. For values 0 (or by not specifying this line), the fugacity coefficients are automatically computed using the Peng-Robinson equation of state. Note the critical pressure, critical temperature, and acentric factor need to be specified in the molecule file.

- `"IdealGasRosenbluthWeight" : floating-point-number`
  The ideal Rosenbluth weight is the growth factor of the CBMC algorithm for a single chain in an empty box. The value only depends on temperature and therefore needs to be computed only once. For adsorption, specifying the value in advance is convenient because the applied pressure does not need to be corrected afterwards (the Rosenbluth weight corresponds to a shift in the chemical potential reference value, and the chemical potential is directly obtained from the fugacity). For equimolar mixtures this is essential.

- `"CreateNumberOfMolecules" : integer`
  The number of molecule to create for the current component. Note these molecules are *in addition* to anything read in by using a restart-file. Usually, when the restart-file is used the amount here should be put back to zero. A warning, putting this value unreasonably high results in an infinite loop. The routine accepts molecules that are grown causing no overlap (energy smaller than 'EnergyOverlapCriteria'). Also the initial starting configurations are far from optimal and substantial equilibration is needed to reduce the energy. However, the CBMC growth is able to reach very high densities.

- `"BlockingPockets" : [[3 x floating-point-number, floating-point-number]]`
  Block certain pockets in the simulation volume. The growth of a molecule is not allowed in a blocked pocket. A typical example is the sodalite cages in FAU and LTA-type zeolites, these are not accessible to molecules like methane and bigger. The pockets are specified as a list of 4 floating points numbers: the s_x, s_y, s_z fractional positions and a radius in Angstrom.

  For example, blocking pockets for ITQ-29 for small molecules are specified as:

```
"BlockingPockets" : [
          [0.0,       0.0,        0.0,       4.0],
```

```
    [0.5,       0.0,        0.0,       0.5],
    [0.0,       0.5,        0.0,       0.5],
    [0.0,       0.0,        0.5,       0.5]
]
```

2.4.2 Component MC-moves

- `"TranslationProbability" : floating-point-number`
  The relative probability to attempt a translation move for the current component. A random displacement is chosen in the allowed directions (see 'TranslationDirection'). Note that the internal configuration of the molecule is unchanged by this move. The maximum displacement is scaled during the simulation to achieve an acceptance ratio of 50%.

- `"RandomTranslationProbability" : floating-point-number`
  The relative probability to attempt a random translation move for the current component. The displacement is chosen such that any position in the box can reached. It is therefore similar as reinsertion, but 'reinsertion' changes the internal conformation of a molecule and uses biasing.

- `"RotationProbability" : floating-point-number`
  The relative probability to attempt a random rotation move for the current component. The rotation is around the starting bead. A random vector on a sphere is generated, and the rotation is random around this vector.

- `"ReinsertionProbability" : floating-point-number`
  The relative probability to attempt a full reinsertion move for the current component. Multiple first beads are chosen, and one of these is selected according to its Boltzmann weight. The remaining part of the molecule is grown using biasing. This move is very useful, and often necessary, to change the internal configuration of flexible molecules.

- `"SwapProbability" : floating-point-number`
  The relative probability to attempt a insertion or deletion move. Whether to insert or delete is decided randomly with a probability of 50% for each. The swap move imposes a chemical equilibrium between the system and an imaginary particle reservoir for the current component. The move starts with multiple first bead, and grows the remainder of the molecule using biasing.

- `GibbsSwapProbability" : floating-point-number`
  The relative probability to attempt a Gibbs swap MC move for the current component. The 'GibbsSwapMove' transfers a randomly selected particle from one box to the other (50% probability to transfer a particle from box I to II, an 50% visa versa).

- `"WidomProbability" : floating-point-number`
  The relative probability to attempt a Widom particle insertion move for the current component. The Widom particle insertion moves measure the chemical potential and can be directly related to Henry coefficients and heats of adsorption.