MoleCuilder - a Molecule Builder

Frederik Heber

MoleCuilder - a Molecule Builder

Frederik Heber

Publication date 07/03/14

Table of Contents

1. Introduction 1
    - What is MoleCuilder? 1
        - Installation requirements 1
        - License 1
        - Disclaimer 1
        - Feedback 2
        - Notation 2
        - Completeness 2
2. Features 3
    - Concepts 3
    - Interfaces 3
    - Known File formats 4
3. Interfaces 6
    - Command-line interface 6
        - Preliminaries 7
        - File parsers 8
        - Selections and unselections 9
        - Shapes 13
        - Geometry Objects 15
        - Randomization 17
        - Manipulate atoms 17
        - Bond-related manipulation 20
        - Manipulate molecules 23
        - Manipulate domain 26
        - Filling 28
        - Analysis 30
        - Fragmentation 32
        - Homologies 37
        - Evaluate Stability 38
        - AtomFragments 38
        - Potentials 39
        - Dynamics 43
        - Tesselations 46
        - Various commands 47
        - Sessions 48
        - Various specific commands 49
    - Text menu 51
    - Graphical user interface 52
        - Basic view 52
        - Selections 54
        - Dialogs 54
    - Python interface 57
4. Conclusions 59
    - Thanks 59

List of Figures

1.1. MoleCuilder logo depicting a tesselated buckyball and a benzene molecule 1
3.1. Screenshot of the basic view of the GUI after loading a file with eight water molecules. 52
3.2. Screenshot of a dialog showing a domain query 55
3.3. Screenshot the add atom action containing an element query 55
3.4. Screenshot of a complex dialog consisting of multiple queries 56
3.5. Screenshort showing the exit dialog 57

List of Tables

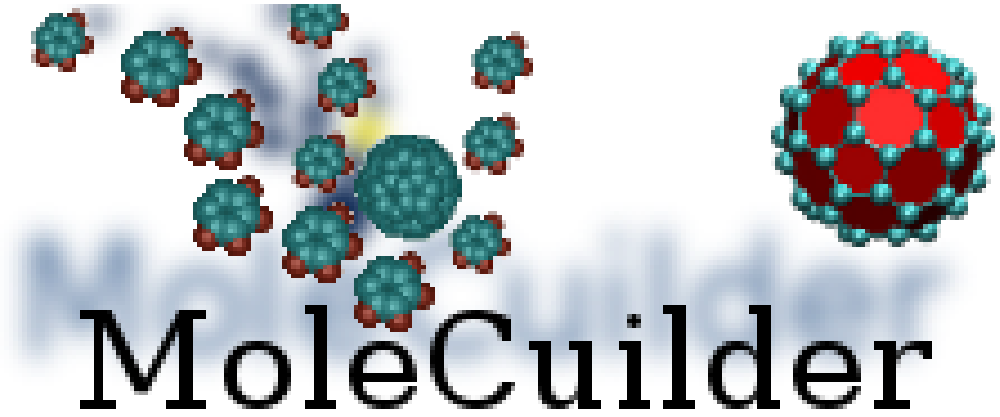
3.1. Keymap 53

List of Equations

3.1. 20

Chapter 1. Introduction

Figure 1.1. MoleCuilder logo depicting a tesselated buckyball and a benzene molecule



What is MoleCuilder?

In Short, **MoleCuilder** is a concatenation of molecule and builder.

In more words, molecular dynamics simulations are frequently employed to simulate material behavior under stress, chemical reactions such as of cementitious materials, or folding pathways and docking procedures of bio proteins. Even if the computational load, due to the large number of atoms, is very demanding, nonetheless they may serve as a starting point, e.g. extracting parameters for a coarser model. However, what is on the other hand the starting point of molecular dynamics simulations? It is the coordinate and element of each atom combined with potential functions that model the interactions.

MoleCuilder allows to fully construct such a starting point: letting the user construct atomic and molecular geometries by a simple point&click approach, a CAD-pendant on the nanoscale. Creating suitable empirical potentials by fitting parameters to ab-initio calculations within hours. Specific emphasis is placed on a simple-to-use interface, allowing for the quick-and-dirty building of molecular systems, and on scriptability. The last being important a eventually not a single, but many, related molecular systems have to be created.

We hope you will enjoy using MoleCuilder as much as we had creating it and still continue extending it. It obtains its flexibility from the use of agile programming techniques and state-of-the-art libraries such as Boost. If you feel dissatiesfied with certain parts, please do not hesitate to give feedback (see below).

Installation requirements

For installations requirements and instructions we refer to the internal documentation of MoleCuilder, created via doxygen™. from the source code.

License

As long as no other license statement is given, MoleCuilder is free for user under the GNU Public License (GPL) Version 2 (see `www.gnu.de/documents/gpl-2.0.de.html`).

Disclaimer

We quote section 11 from the GPLv2 license:

Because the program is licensed free of charge, there is not warranty for the program, to the extent permitted by applicable law. Except when otherwise stated in writing in the copyright holders and/or other parties provide the program "as is" without warranty of any kind, either expressed or implied. Including, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. The entire risk as to the quality and performance of the program is with you. Should the program prove defective, you assume the cost of all necessary servicing, repair, or correction.

Feedback

If you encounter any bugs, errors, or would like to submit feature request, please use the email address provided at the very beginning of this user guide. The author is especially thankful for any description of all related events prior to occurrence of the error, saved "session scripts" (see below) and auxiliary files. Please mind sensible space restrictions of email attachments.

Notation

We briefly explain a few specific wordings associated with the program:

- *Action* is a command that allows for undoing and redoing, i.e. a single atomic procedure for manipulating the molecular system. Here, atomic refers to indivisible and not to atoms. It is also referred to as a command.
- Selection refers to a subsets from the set of instances of a particular type, e.g. atoms, molecules, shapes, ...
- Shape means a specific region of the domain that can be described in the way of constructive geometry, i.e. as the intersection, negation, and combination of primitives such as spheres, cubes, or cylinders.
- World refers to the whole of the molecular system, i.e. all atoms with coordinates and element type (over all time steps), all bonds between pairs of atoms, the size of the simulation domain. This is also referred to as the state.
- Time step is the current discrete position in time. Molecular dynamics simulations are executed in discrete (but very small) time steps. Each atom has a distinct position per time step. The discrete positions over the discrete time steps samples its trajectory during a simulation.

Completeness

This documentation takes quite some effort to write. Hence, the described features and especially the actions herein are settled with respect to their functionality, while newer features or actions are probably missing. This should be a clear sign to you that these are probably not safe to use yet. If you nonetheless require them, you should acquire some familiarity with the code itself. This suggests changing to the developer documentation which is maintained along with the source code with doxygen™.

Chapter 2. Features

Basically, **MoleCuilder** parses geometries from files, manipulates them, and stores them again in files. The manipulation can be done either via a command-line interface or via the graphical user interface.

Concepts

In general, we divide the molecular systems into three different components or scales.

1. Atoms

   Atoms are the undividable objects of the molecular systems. They have at least an element “Z” and three coordinates “(x,y,z)”.

2. Molecules

   Molecules are bound conglomeration of atoms. They contain a number of atoms, i.e. by beginning at an arbitrary atom of the molecule and traversing its bond graph eventually all of the molecule's atoms are visited. Currently, a bond refers to a covalent bonding between atoms. Also, molecules may have a bounding box, i.e. a subdomain that contains all of the atoms in the molecule.

   Note that the molecular structure of the system, i.e. the bonding graph, is determined by MoleCuilder and used to dissect the system into distinct molecules on request.

3. Clusters

   Clusters are unbound conglomeration of atoms. Clusters serves as groups of atoms for specific operations that would be to restricted if they worked on just molecules.

4. Domain

   The domain refers to the simulation domain. It is the parallelepiped in where either periodic, wrapped, or open boundary conditions apply. The domain contains all atoms, i.e. the box containing all atoms.

Interfaces

MoleCuilder has four different interfaces: Command-line, text menu, graphical user interface, and python interface.

1. Command-Line

   The command-line interface allows to use MoleCuilder non-interactively via a terminal session. The program is executed by appending to the shell command a number of commands including all required options that are then executed one after the other. After execution of the last command, the program quits. The command-line interface usually works on a specific file that is given as input, manipulated, analysed, ... via the sequence of commands and eventually all changes are stored in the this file. Hence, the input file acts as the state of the starting configuration that is modified via MoleCuilder.

2. Text menu

   The text-menu is similar to the command-line interface with the exception that it allows for interactive sessions. Commands are chosen from a text menu and executed directly after selection by the user.

3. Graphical interface

The graphical interface is based on Qt. It features a full graphical, three-dimensional representation of the simulation domain with atoms and their bonds. It allows manipulation in point&click fashion. Commands are selected from pull-down menus and dialogs are used to query the user for all required parameters to such a command.

4. Python interface

   The last interface is accessible only within the python programming language. MoleCuilder can be loaded as a module and its commands can be executed with either the python interpreter interactively or via python scripts non-interactively. Note that this allows auxiliary calculations to be performed in pythons whose results may be used as parameters in subsequent commands.

Known File formats

We briefly list the file formats MoleCuilder can parse and store. We refer to external websites for more detailed information where appropriate.

- XYZ, `.xyz` (simplest of all formats, line-wise element and three coordinates with two line header, number of lines and a comment line)
- MPQC ™ [http://www.mpqc.org/], `.in`
- PDB [http://www.pdb.org/], `.pdb`
- ESPACK™, `.conf` (electronic structure package by Institute for Numerical Simulation, University of Bonn, code not in circulation)
- PSI4 ™ [http://www.psicode.org/], `.psi`
- TREMOLO™ [http://www.tremolo-x.org/], `.data`
- XML, `.xml` (XML as read by ScaFaCoS [http://www.scafacos.org/] project)

These are identified via their suffixes and can be converted from one into another (with possible loss of data outside of the intersection of stored properties of the two involved file formats).

Note that the various formats differ in how much information they can store. This restriction may come from two sources: either the file format specification does not support more or the implementation is still lacking specific features.

What all formats can store at least is the element per atom and a position in three spatial coordinates for a single time step.

In the following we list some restrictions:

- Multiple time steps supported by
  - Pdb
  - Tremolo (note entries allows to change over steps are: x, u, F)
  - Xyz
- Bond information supported by
  - Pdb
  - Tremolo
- Full atom state storable (i.e. position, velocity, and force)

  - Tremolo

- Domain information (however, this is not parsed only stored)

  - Tremolo

You easily notice that the Tremolo™ format is currently the most versatile. However, by advised that you need to give a specific *ATOMINFO* line that specifies what needs to be stored, see set-tremolo-atomdata.

Chapter 3. Interfaces

In this chapter, we explain the intention and use of the four interfaces.

We will give the most extensive explanation of the command-line interface, all subsequent interfaces are explained in highlighting their differences with respect to the command-line interface. This is because the command-line lends itself very well to representation in this textual user guide. Although some images of the graphical interface are given below, they would blow the size of the guide out of proportion.

In any case, you should make yourself familiar with at least one of the interactive (text menu, GUI) and one of the non-interactive (command-line, python) interfaces to use MoleCuilder to its full potential: The interactive interface gives you the immediate feedback in constructing "synthesis" (build) chains (of commands) for constructing your specific molecular system in the computer. The non-interactive interface lends itself to quick creation of related systems that differ only by specific parameters you have modified in the script (command-line can be used in shell scripts, python itself is a scripted language). Also, the non-interactive interfaces are used for storing sessions which helps you in documentation your experiments and later on understanding of what has been actually created by the prescribed commands, i.e. debugging.

Command-line interface

The command-line interface reads options and commands from the command line and executes them sequentially. This may be for example: Open an empty file, add 2 hydrogen atoms and add 1 oxygen atom, recognize the bond graph, choose a simulation box, fill the box with this given "filler" molecule, save the file. This enables the use of MoleCuilder in simple script-files to create a whole range of geometries that only differ in a few parameters automatically.

Traditionally, **MoleCuilder** operates on a single configuration file - the state - which may also store additional information depending on the chosen file format such as parameters for ab-initio computations. To some small extent **MoleCuilder** also allows manipulation of these paramters. An example for the above procedure is given below:

```
./molecuilder \
 -i sample.xyz \
 --add-atom H \
   --domain-position "0.,0.,0." \
 ...
```

The first argument is the executable itself. Second, there is a slew of arguments -- one per line split with a backslash telling the comman shell that the line still continues -- consisting of the input action and an arbitrarily named file `sample.xyz`, which may be empty and whose file format is chosen by the given extension. The third is the add-atom action following by an option that gives the position in the domain where to add the "H"ydrogen atom. An action is always introduced via a double hyphen and its full name (containing just non-capital letters and hyphens) or a single hyphen and a single letter for its shortform, such as **-a** for adding an atom to the system. It is followed by a fixed number of options. Most of these have default values and in this do not have to be specified.

Invalid values. Certain options accept only very specific input. For example, the option value associated with **add-atom**, i.e. the chemical element, can only be one of the chemical symbols that exist and not any arbitrary string. Each option value is checked on parsing the command-line. If any value is not valid, an error message is given and none of the actions is executed.

Shortforms of Actions. Note that not all action have shortforms and it is best practice to have the full action name instead of its shortform to make the command-line comprehendable to you in years to come.

Note

Note further that when placing a slew of commands in a script file it is generally recommended to use the above formatting: One command or option per line and each receives an extra tab for indentation.

Preliminaries

Some preliminary remarks are in order which we have gathered here on how these actions work in general.

We first delve into some details about secondary structure such as selections, shapes, and randomization required to specify subsets of atoms and molecules you wish to manipulate. Then, we have give the details on the manipulation ordered by the scale they act upon - single atoms, multiple atoms organized as molecules, and all atoms organized by their containing domain.

In the following we will always give a command to illustrate the procedure but just its necessary parts, i.e. "..." implies to prepend it with the executable and input command for a specific configuration file, for storing the manipulated state of the molecular system.

So if we write

```
... --help
```

Then we actually mean you to write

```
./molecuilder --help
```

Note that this specific exemplary command is very useful as it will always give you a list of all available actions and also a brief explanation on how to properly enter values of a specific type, e.g. an element, a vector, or a list of numbers. Details to a specific action can be requested when its full name is known, e.g. for "add-atom",

```
./molecuilder --help add-atom
```

which fills you in on each option to the action: its full name, its expected type, and a possibly present default value, and a brief description of the option.

An Action can be undone and redone, e.g. undo adding an atom as follows,

```
... --add-atom H --domain-position "0,0,0" --undo
```

and redo as follows

```
... --add-atom H --domain-position "0,0,0" --undo --redo
```

With the non-interactive interfaces this may seem rather superfluous but it comes in very handy in the interactive ones. Also this should tell you that actions are placed internally in a queue, i.e. a history, that undo and redo manipulate.

Furthermore, a specific event in the history of actions can be marked using

```
... --add-atom H --domain-position "0,0,0" --set-undo-mark 1 --select-all-atoms
```

Using this mark (that is unset by

```
--set-undo-mark 0
```

) one can then step directly back to that event by undoing all actions in between the current and the marked event.

```
--undo-till-mark
```

Here, this will jump back to when the hydrogen atom was just added. This function is required as not all actions produce an event in the Action history. Certain events will just calculate something and

output, i.e. they do not change the state. However, thereby one cannot simply count actions to return back to a specific state. Hence, this function allows to mark a specific state beforehand.

Due to a current limitation of the implementation each command can be used on the command-line only once. Note that this *only* applies to the command-line interface. All other interfaces, especially all interactive ones, do not have such a restriction. For the command-line interface there are several ways to work around it. Either by splitting the whole chain of commands into several chunks, each using only unique commands and using the input file (the state) to contain and transport the intermediate stages as input for the next stage. Or to switch to other commands: often there are several possible ways of achieving a goal, especially when using selections.

Being done now with the preliminaries we now go through all available actions present in MoleCuilder.

File parsers

We have already given a list of all known file formats, see File formats. Next, we explain how these file formats are picked and manipulated.

Parsing files

We already discussed that the command-line interface works state-based and hence you should supply it with a file to work on.

```
... --input water.data
```

This will load all information, especially atoms with their element and position, from the file `water.data` into the state. Most importantly, all changes will eventually be stored to this file, or to files with the prefix `water` and suffixes of desired file formats, e.g. `water.in` if you specified MPQC™.

```
... --load morewater.xyz
```

This will load another file `water.xyz`, however changes will still be written to files prefixed with `water` as designated by the **input** command. Note that now already two state files will stored, `water.data` and `water.xyz` as these two different file formats have been used. This is the default behavior: any additional file format used in loading is registered internally and the output file will then be written in all registered formats on exit.

Note

If the loaded file is empty, then no parser is registered. This means if a new state file needs to be written,then the output format has to be stated explicitly, e.g.

```
... --input hydrogen.xyz \
    --set-output xyz \
    --add-atom H --domain-position "0,0,0"
```

Note

In the case that parsing all time steps from a given input file will take too long, especially for larger systems, fast parsing may be activated, only the first time step is loaded, all other are ignored.

```
... --fastparsing 1
```

Adding output file formats

We already know that loading a file also picks a file format by its suffix. We may add further file formats to which the state of the molecular system is written to on program exit.

```
... --set-output mpqc tremolo
```

This will store the final state of the molecular systems as MPQC™ and as TREMOLO™ configuration file. See File formats for the list of all file formats available.

Storing to files

Storing the current World, i.e. all its atoms, to a given file, happens automaticallyon program exists if an input file has been specified. However, this can also be forced at any point in between by using one of two actions:

```
... --output
```

This action does not use an argument and will simply use the currently registered input file and store the state of the World there This is handy if an intermediate state is required (making sense for interactive and python interfaces).

```
... --output-as world.xyz
```

This action on the other hand will be write the current state to a new file "world.xyz".

By default, this will not overwrite any present files. To overwrite present files, please append

```
--force-overwrite 1
```

.

Output the current molecular system

This will store all atoms contained in the currently selected molecules to file. This is different to**store-saturated-fragment** as it will not saturate dangling bonds because only whole molecules, i.e. whose bond graph is connected, will be stored.

```
... --save-selected-molecules waters.pdb
```

Selections and unselections

In order to tell MoleCuilder on what subset of atoms a specific Action is to be performed, there are *selection actions*. Note that a selection per se does not change anything in the state of the molecular system in any way. Essentially, it is just a filter.

Selections either work on atoms, on molecules, or on shapes (this we explain later on). A given selection is maintained from the execution of the selection action to the end of program or until modified by another selection applied on the same type (atom, molecule, shape). Selections are not stored to file (i.e. do not belong to the state).

We only give here a brief list on the available kind of selections for each of the three types they work on. Each action is executed either as follows, exemplified by selecting all atoms.

```
.... --select-all-atoms
```

or, exemplified by unselecting the last molecule,

```
... --unselect-molecule-by-order -1
```

i.e. they are prepended by either **select** or **unselect**.

- Atoms
  - All

```
... --select-all-atoms
```

  - None

```
... --unselect-all-atoms
```

```
... --clear-atom-selection
```

- Invert selection

```
... --invert-atoms
```

- By Element (all hydrogen atoms, all sulphur atoms, ...)

```
... --select-atom-by-element 1
```

```
... --unselect-atom-by-element 1
```

- By Bond Neighbors (all bond neighbors of all currently selected atoms are additionally selected)

```
... --select-atom-bond-neighbors
```

- By name (atom with molecule internal name "H1", e.g. this will select all "H1" atoms in every present water molecule)

```
... --select-atom-by-name "H1"
```

```
... --unselect-atom-by-name "H1"
```

- By Id (atom with id 76)

```
... --select-atom-by-id 76
```

```
... --unselect-atom-by-id 76
```

- By Order (the first (1), the second, ... the last created(-1), the last but one)

```
... --select-atom-by-order 1
```

```
... --unselect-atom-by-order -2
```

- By Random Number of Atoms (e.g., 5 random atoms)

```
... --select-atom-by-random 5
```

- By Shape (all atoms inside the volume specified by the currently selected shape)

```
... --select-atom-inside-volume
```

```
... --unselect-atoms-inside-volume
```

- By Molecule (all atoms belonging to currently selected molecules)

```
... --select-molecules-atoms
```

```
... --unselect-molecules-atoms
```

- Push/Pop the current selection to/from a stack to store it momentarily and allow modifications in MakroActions (this is very specific and used mostly internally).

```
... --push-atom-selection
```

```
... --pop-atom-selection
```

- Select bond neighbors of all currently selected atoms.

```
... --select-atoms-neighbors
```

- Molecules
  - All

```
... --select-all-molecules
```

  - None

```
... --unselect-all-molecules
```

```
... --clear-molecule-selection
```

- Invert selection

```
... --invert-molecules
```

- By Id (molecule with id 4)

```
... --select-molecule-by-id 2
```

```
... --unselect-molecule-by-id 2
```

- By Order (first created molecule, second created molecule, ...)

```
... --select-molecule-by-order 2
```

```
... --unselect-molecule-by-order -2
```

- By Formula (molecule with H2O as formula)

```
... --select-molecules-by-formula "H2O"
```

```
... --unselect-molecules-by-formula "H2O"
```

- By Name (all molecules named "water4")

```
... --select-molecules-by-name "water4"
```

```
... --unselect-molecules-by-name "water4"
```

- By Atom (all molecules for which at least one atom is currently selected)

```
... --select-atoms-molecules
```

```
... --unselect-atoms-molecules
```

- Push/Pop the current selection to/from a stack to store it momentarily and allow modifications in MakroActions.

```
... --push-molecule-selection
```

```
... --pop-molecule-selection
```

- Shapes
  - All

```
... --select-all-shapes
```

  - None

```
... --unselect-all-shapes
```

  - By Name (all shapes named "sphere1")

```
... --select-shape-by-name "sphere1"
```

```
... --unselect-shape-by-name "sphere1"
```

Note that an unselected instance (e.g. an atom) remains unselected upon further unselection and vice versa with selection.

These above selections work then in conjunction with other actions and make them very powerful, e.g. you can remove all atoms inside a sphere by a selecting the spherical shape and subsequently selecting all atoms inside the shape and then removing them.

Shapes

Shapes are specific regions of the domain. There are just a few so-called *primitive* shapes such as cuboid, sphere, cylinder, the whole domain, or none of it. However, these can be combined via boolean operations such as and, or, and not. This approach is called *constructive geometry*. E.g. by combining a sphere with the negated (*not* operation) of a smaller sphere, we obtain a spherical surface of specific thickness.

> **Note**
> Shapes are especially useful in the QtGui where all selected shapes are visualized by their translucent boundary. This makes it especially easy to e.g. recognize all atoms that would be removed after selecting all atoms inside the volume of a selected shape.

Creating shapes

Primitive shapes can be created as follows,

```
... --create-shape \
    --shape-type sphere \
    --shape-name "sphere1" \
```

```
    --stretch "2,2,2" \
    --translation "5,5,5"
```

This will create a sphere of radius 2 (initial radius is 1) with name "sphere1" that is centered at (5,5,5). Other primitives are cuboid and cylinder, where a rotation can be specified as follows.

```
... --create-shape \
    --shape-type cuboid \
    --shape-name "box" \
    --stretch "1,2,2" \
    --translation "5,5,5" \
    --angle-x "90"
```

```
... --create-shape \
    --shape-type cylinder \
    --shape-name "cylinder" \
    --stretch "1,2,2" \
    --translation "5,5,5" \
    --angle-y "90"
```

Combining shapes

Any two shapes can be combined by boolean operations as follows

```
... --combine-shapes \
    --shape-name "combinedshape" \
    --shape-op "AND"
```

This will combine two currently selected shapes vis the "AND" operation and create a new shape called "combinedshape". Note that the two old shapes are still present after this operation. We briefly explain each operation:

- *AND* combines two currently selected shapes into a new shape that consists of only the volume where shapes overlap.
- *OR* combines two currently selected shapes into a new shape that consists of all the volume that either shape occupies.
- *NOT* creates the inverse to a currently selected single shape that contains the volume with respect to the simulation domain that the present one does not.

Removing shapes

Removing a shape is as simple as removing an atom.

```
... --remove-shape
```

This removes all currently selected shapes.

Manipulating shapes

Shapes can be stretched, scaled, rotated, and translated to modify primitives or combined primitive shapes. As you have seen this manipulation could have occurred already at creation but we may also

do it later on. As usual, we just list examples of the various manipulations below, each of them works on the currently selected shapes.

```
... --stretch-shapes "1,1,2" \
    --stretch-center "5,5,5"
```

This stretches the shapes relative to the center at (5,5,5) (default is origin) by a factor of 2 in the z direction.

```
... --rotate-shapes \
    --center "10,2,2" \
    --angle-x 90 \
    --angle-y 0 \
    --angle-z 0
```

This way all selected shapes are rotated by 90 degrees around the x axis with respect to the center at (10,2,2).

```
... --translate-shapes "5,0,0"
```

This translates all selected shapes by 5 along the x axis.

Geometry Objects

Although we use the term geometry objects in the title, we actually mean vectors, i.e. a position or direction in the three-dimensional space. But maybe we have need for the more general term in the future.

Vectors are required as input to many of the Actions further below: translating atoms, rotating atoms around a specific axis, aligning a molecule with a vector, ...

Therefore, vectors can be stored and referenced using a given name. This allows for a very powerful and handy manipulation of the molecular system afterwards. And to give a concrete example, let's have a look at translating a set of selected atoms, see subsection on Translating atoms.

```
... --translate-atoms "unitVectorX"
```

This would use the automatically created reference "unitVectorX", i.e. the vector with components (1,0,0) as the translation vector for the given set of atoms. In other words, all selected atoms get shifted by 1 unit (e.g. Angström) in +X direction.

We have the following automatically created geometry objects whose names are self-explanatory:

- zeroVector
- unitVectorX
- unitVectorY
- unitVectorZ

However, more vectors can be simply constructed from atomic positions, such as the position of an atom directly, the distance between two atoms (in case they are bonded, then this would be the bond vector) or from three atoms, defining a plane and giving its normal vector.

We have refrained from giving automated names to vectors and even keeping them up-to-date automatically, i.e. the distance between two atoms O1 and O2 could be named "distance_O1_O2" or similar. However, we want the user to have full control and maybe come up with more suitable names such as "rotation_axis" in this case.

Warning

Note that names have to be unique and the Action will fail if the name is already used.

Atomic distance to stored vector

The distance between two selected atoms is stored as a vector as follows,

```
... --distance-to-vector "distance_vec" \
    --reverse 0
```

where the distance vector can be referenced by "distance_vec" from then on in other Actions requiring a vector as input.

Note

Since selected atoms are used in the fixed order of their ids (and not in the order they were clicked at in QtGui), the direction can be inverted by giving the "reverse" option. It is off (or 0) by default.

Coordinates to stored vector

We may also create a geometry vector simply by supplying the three coordinates of a vector.

```
... --input-to-vector "vector" \
    --position "1,2,3"
```

where the vector with components (1,2,3) can be referenced by "vector" .

Normal of plane to stored vector

Three positions in space (if they are not linear dependent) define a plane in three-dimensional space.

Therefore, when exactly three atoms are selected, this Action will construct the resulting plane and store its normal vector as a geometry object for later reference.

```
... --plane-to-vector "planenormal" \
    --reverse 1
```

where the plane's normal vector can be referenced by "planenormal". The additional "reverse" option will invert the plane's normal vector as this is ambiguous about the direction.

Atomic position to stored vector

Storing the position of a singly selected atom as a vector is simply done as follows,

```
... --position-to-vector "vector_O1" \
```

where the vector can be referenced by "vector_O1" from then on.

Remove a stored vector

Finally, a stored vector can also be removed.

```
... --remove-geometry "vector_O1" \
```

this removes the stored "vector_O1".

Randomization

Some operations require randomness as input, e.g. when filling a domain with molecules these may be randomly translated and rotated. Random values are obtained by a random number generator that consists of two parts: engine and distribution. The engine yields a *uniform* set of random numbers in a specific interval, the distribution modifies them, e.g. to become gaussian.

There are several Actions to modify the specific engine and distribution and their parameters. One example usage is that with the aforementioned filling of the domain (see below) molecules are rotated randomly. If you specify a random number generator that randomly just spills out values 0,1,2,3, then the randomness is just the orientation of the molecule with respect to a specific axis: x,y,z. (rotation is at most 360 degrees and 0,1,2,3 act as divisor, hence rotation angle will then be always a multiple of 90 degrees).

```
... --set-random-number-distribution "uniform_int" \
    --random-number-distribution-parameters "p=1"
```

This changes the distribution to "uniform_int", i.e. integer numbers that are distributed uniformly.

```
... --set-random-number-engine "mt19937" \
    --random-numner-engine-parameters "seed=10"
```

Specifying the seed allows you to obtain the same sequence of random numbers for testing purposes.

Moreover, actions such as random perturbation of atoms and filling of molecules into the domain use the random number generator.

Manipulate atoms

Here, we explain in detail how to add, remove atoms, change its element type, scale the bond in between or measure the bond length or angle.

Adding atoms

Adding an atom to the domain requires the element of the atom and its coordinates as follows,

```
... --add-atom O \
    --domain-position "2.,3.,2.35"
```

where the element is given via its chemical symbol and the vector gives the position within the domain

Note that instead of giving an explicit vector you may also use a vector stored as a geometry object, see section Geometry.

Finally, if you select a molecule before adding the atom, then the atom is added to that particular molecule. Otherwise a new molecule is created internally to contain that atom. Atoms always need to be associated with a molecule.

Randomly perturb atoms

The positions of a set of selected atom(s) can be randomly perturbed by giving a maximum noise level..

```
... --random-number-distribution "uniform_01" \
    --random-perturbation 0.1
```

This will perturb all atomic positions by adding a vector with components chosen randomly from the interval [-level, level].

Note that the manner these random numbers are picked depends on the current random number distribution (and also on the engine), see Randomization. The default one will not give good results, therefore the example uses the uniform_01 distribution

Removing atoms

Removing atom(s) does not need any option and operates on the currently selected ones.

```
... --remove-atom
```

Saturating atoms

Newly instantiated atoms have no bonds to any other atom. If you want to fill up their valence by a slew of hydrogen atoms residing on a sphere around this atom, use this action.

```
  ... --saturate-atoms
```

A number of hydrogen atoms is added around each selected atom corresponding to the valence of the chemical element. The hydrogen atoms are placed at the same distance to the selected atom which is taken from an internal database for the typical hydrogen bond length and approximately with same distance to their nearest neighbor hydrogens. Already present bonds (i.e. the position of neighboring atoms) are taken into account and left intact.

```
^M
  ... --saturate-atoms --use-outer-shell 1^M
```

This will not simply consider the number of unoccupied orbitals of the respective element. Instead it will take the outermost shell into consideration and use the number of orbitals in total as the ideal polyhedra but only taking the number of unoccupied orbitals as free nodes, i.e. where to saturate with hydrogens. For example, nitrogen will then fill an tetrahedron (4 orbitals in outer shell) where one orbital is already fully occupied and 3 nodes in the tetrahedron remain to fill up with hydrogens. Without the outer shell, saturate atoms would simply consider a triangle as the ideal polyhedron and fill its 3 corners.

Bonding atoms

This action is to an extent the opposite of saturate-atom. Instead of adding hydrogens to saturate an atom with unoccupied valence orbitals MoleCuilder looks for other non-hydrogen atoms in the vicinity that have bonds with hydrogen atoms. If these non-hydrogen atoms are within suitable bonding

distance, the hydrogen bond with the hydrogen atom is removed and a bond between the two non-hydrogen atoms is added. Hence, while saturate-atom adds more hydrogens, bondify-atom will try to remove hydrogens and saturate atom valencies by additional bonds.

> **Note**
> The typical bond distances between the two non-hydrogen atoms is taken from bond table, loaded through bond-table.

The action is called as follows:

```
... --bondify-atoms
```

and will work on a single, currrently selected atom.

Translating atoms

In order to translate the current selected subset of atoms you have to specify a translation vector.

```
... --translate-atoms "-1,0,0" \
    --periodic 0
```

This translates all atoms by "-1" along the x axis and does not mind the boundary conditions, i.e. it might shift atoms outside of the domain.

Again, note that instead of giving an explicit vector you may also use a vector stored as a geometry object, see section Geometry.

Mirroring atoms

Present (and selected) atoms can be mirrored with respect to a certain plane. You have to specify the normal vector of the plane and the offset with respect to the origin as follows

```
... --mirror-atoms "1,0,0" \
    --plane-offset 10.1 \
    --periodic 0
```

And of course instead of giving an explicit vector you may also use a vector stored as a geometry object, see section Geometry.

Translating atoms to origin

The following Action is convenient to place a subset of atoms at a known position, the origin, and then translate them to some other absolute coordinate. It calculates the average position of the set of selected atoms and then translates all atoms by the negative of this center, i.e. the center over all selected atoms is afterwards at the origin.

```
... --translate-to-origin
```

Note that this naturally does not heed the boundary conditions of the simulation domain.

Changing an atoms element

You can easily turn lead or silver into gold, by selecting the silver atom and calling the change element action.

```
... --change-element Au
```

Bond-related manipulation

Atoms can also be manipulated with respect to the bonds. *Note that with bonds we always mean covalent bonds.* First, we explain how to modify the bond structure itself, then we go in the details of using the bond information to change bond distance and angles.

Creating a bond graph

In case you have loaded a configuration file with no bond information, e.g. XYZ, it is necessary to create the bond graph. This is done by either of two options, based on a heuristic criterion.

By default it is based on van-der-Waals radii, i.e. if we look at two atoms indexed by "a" and "b"

Equation 3.1.

where is the lookup table for the radii for a given element and \tau is a threshold value, set to 0.4.

As a alternative option, you may load a file containing bond table information.

```
... --bond-table table.dat
```

which would parse a file `table.dat` for a table giving typical bond distances between elements a and b. These are used in the above criterion as in place of .

In either case,

```
... --create-adjacency
```

will then create the bond graph based on above criterion.

Destroying the bond graph

The bond graph can be removed completely (and all bonds along with it).

```
... --destroy-adjacency
```

Correcting bond degrees

Typically, after loading an input file with bond information, e.g. a PDB file, the bond graph is complete but we lack the weights. That is we do not know whether a bond is single, double, triple, ... This action corrects the bond degree by enforcing charge neutrality among the connected atoms.

This action is in fact quadratically scaling in the number of atoms. Hence, for large systems this may take longer than expected.

```
... --correct-bonddegree
```

However, in normal use scenarios the action is fast and linear scaling.

Analysing a bond graph

You can perform a depth-first search analysis that reveals cycles and other graph-related information.

```
... --depth-first-search
```

Note that this will only print some information and has no other impact on the state.

Dissecting the molecular system into molecules

The bond graph information can be used to recognize the molecules within the system. It removes the current bond graph (if any bonds are presents) and recreates it. Afterwards it walks it along and finds any disconnected subgraphs, associating a molecule with each. Note that this is just structural information and does not change the state of the system.

```
... --subgraph-dissection
```

Updating molecule structure

When the bond information has changed, new molecules might have formed, this action updates all the molecules by scanning the connectedness of the bond graph of the molecular system.

Moreover, Imagine you have just loaded a PDB file containing bond information. However, initially all atoms are dumped into the same molecule. Before you can start manipulating, you need to dissect the system into individual molecules.

```
... --update-molecules
```

Adding a bond manually

When the automatically created adjacency or bond graph contains faulty bonds or lacks some, you can add them manually.

```
... --add-bonds
```

If two atoms are selected, the single bond in between is added, if not already present. If more than two atoms are selected, than the bond between any pair of these is added.

Note

This is especially useful in conjunction with the fragmentation scheme (explained later on). If you want to know the contribution from certain fragments whose subgraph is not connected, you can simply make the associated subset of atoms connected by selecting all bonds and adding the bonds.

Removing a bond manually

In much the same way as adding a bond, you can also remove a bond.

```
... --remove-bonds
```

Similarly, if more than two atoms are selected, then all bonds found between any pair of these is removed.

Setting the bond degree manually

The bond degrees are usually automatically set to fulfill the valency constraints of each bond partner. However, degrees can also be set manually for a set of selected atoms. Note that the degree is set to the given value for all bonds in between any pair of atoms within the set.

```
... --set-bond-degree 2
```

Similarly, if more than two atoms are selected, then all bonds found between any pair of these are modified.

Saving bond information

Bond information can be saved to a file in TREMOLO ™'s dbond style.

```
... --save-bonds system.dbonds
```

Similarly is the following Action which saves the bond information as a simple list of one atomic id per line and in the same line, separated by spaces, the ids of all atoms connected to it.

```
... --save-adjacency system.adj
```

This corresponds to the **bond-file** Action.

Load extra bond information

The reverse Action of **save-bonds** is the following which loads bond information from the file **water.dbond**.

```
... --bond-file water.dbond
```

Note that because of the use of ids the bond file is intimately connected to the associated input file containing the atomic coordinates and thus both should parsed right after another and to the very beginning of any sequence of Actions.

Stretching a bond

Stretching a bond actually refers to translation of the associated pair of atoms. However, this action will keep the rest of the molecule to which both atoms belong to invariant as well.

```
... --stretch-bond 1.2
```

This scales the original bond distance to the new bond distance 1.2, shifting the right hand side and the left hand side of the molecule accordingly.

Warning

this fails with aromatic rings (but you can always undo).

Note

The action will uses typical bond distances, taken from bond-table to scale the bond length if the desired bond length is non-positive.

Changing a bond angle

In the same way as stretching a bond, you can change the angle in between two bonds. This works if exactly three atoms are selected and two pairs are bonded.

```
... --change-bond-angle 90
```

This will change the angle from its value to 90 degrees by translating the two outer atoms of this triangle (the atom connected to both other atoms serves as rotation joint).

Evaluate the Chemical Space

Imagine that we are given a graph consisting of nodes and edges. As we have been speaking extensively of the adjacency graph before, then any graph where nodes are assigned a chemical element and edges are given a degree, effectively represent a molecule of covalently bonded atoms.

This is the notion behind constructing parts of the chemical space that encompasses all molecules that either already exist or could be devised, i.e. that are stable.

One method of creating points in this chemical space, i.e. stable molecules, is presented in the article by Hamaekers et al. 2017, where they follow the well-known *octet rule*.

In this action essentially the same method is implemented. Given an arbitrary graph, encoded as Graph6 string, such as produced by the nauty [http://users.cecs.anu.edu.au/~bdm/nauty/] toolset (Brendan McKay and Adolfo Piperno), and a set of chemical elements, the action creates every possible molecule using the graph and the given elements for each of its nodes, changing bond degrees as long as to fulfil the octet rule.

```
--evaluate-chemical-space \
  --graph6 "B`" --elements C C
```

The graph6 string "B`" represents the simplest graph consisting of just two nodes, connected by a single edge. Designating each node to be a carbon atom, this action will then produce the carbonhydrates C2H6, C2H4, and C2H2.

At the moment, the molecules are not created but their signatures are used in looking up the fragments in the internal homology container, see Fragmentation and Homologies. Instead of creating atomic coordinates, approximate energies are given that allow to estimate whether the molecule candidate is stable or not. Note that this requires that a suitable homology container file has been loaded first that contains all necessary fragments. If multiple entries and thus multiple energies are present for a given fragment, then the lowest energy is taken for the summation.

Note

The graph is automatically saturated with hydrogens in obeying the octet rule, i.e. hydrogen should not be given in the list of elements.

Using nauty that produces all possible graphs with a fixed number of nodes, these can be simply fed into MoleCuilder ™ to produce all associated molecules.

Print selected atoms as graph6 string

In the light of the last section it would also be useful to obtain the graph6 representation of a present set of atoms, e.g., those belonging to a molecule. The following command will print the graph6 string to the console for all currently present atoms.

```
--select-all-atoms \
  --print-selected-atoms-as-graphstring
```

Manipulate molecules

Molecules are agglomerations of atoms that are (covalently) bonded. Hence, the Actions working on molecules differ from those working on atoms. Joining two molecules can only be accomplished by adding a bond in between, and in the reverse fashion splitting a molecule by removing some or even all bonds in between. The Actions below mostly deal with copying molecules. Removing of molecules is done via selecting the molecule's atoms and removing them, which removes the atoms as well.

Note

Initially when you load a file via the input action all atoms are placed in a single molecule despite any present bond information, see Dissecting the molecular system into molecules

Copy molecules

A basic operation is to duplicate a molecule. This works on a single, currently selected molecule. Afterwards, we elaborate on a more complex manner of copying, filling a specific shape with molecules.

```
... --copy-molecule \
    --position "10,10,10"
```

This action copies the selected molecule and inserts it at the position (10,10,10) in the domain with respect to the molecule's center. In effect, it copies all the atoms of the original molecule and adds new bonds in between these copied atoms such that their bond subgraphs are identical.

Here, instead of giving an explicit vector you may also use a vector stored as a geometry object, see section Geometry.

Change a molecules name

You can change the name of a molecule which is important for selection.

```
... -change-molname "test
```

This will change the name of the (only) selected molecule to "test".

Connected with this is the default name an unknown molecule gets.

```
... --default-molname test
```

This will change the default name of new molecules to "test".

Note

Note that a molecule loaded from file gets the filename (without suffix) as its name.

Remove molecules

This removes one or multiple selected molecules.

```
... -remove-molecule
```

This essentially just removes all of the molecules' atoms which in turn also causes the removal of the molecule.

Translate molecules

This translates one or multiple selected molecules by a specific offset..

```
... -translate-molecules
```

As before, this is actually just an operation on all of the molecule's atoms, namely translating them.

Same as with translate-atoms instead of giving an explicit vector you may also use a vector stored as a geometry object, see section Geometry.

Rotate around bond

This rotates parts of a molecule around a given bond, i.e. the bond vector becomes the rotation axis but only atoms on the side of second atom get rotated. This naturally does not work for bonds in a cycle.

```
... --rotate-around-bond "90" \
```

```
    --bond-side 0
```

Rotate around self

You can rotate a molecule around its own axis.

```
... --rotate-around-self "90" \
    --axis "0,0,1"
```

This rotates the molecule around the z axis by 90 degrees as if the origin were at its wn center of origin.

Rotate around origin

In the same manner the molecule can be rotated around an external origin.

```
... --rotate-around-origin 90 \
    --position "0,0,1"\
```

This rotates the molecule around an axis from the origin to the position (0,0,1), i.e. around the z axis, by 90 degrees, where for the position you may also use a stored vector, see section Geometry.

Rotate to principal axis system

The principal axis system is given by an ellipsoid that mostly matches the molecules shape. The principal axis system can be simply determined by

```
... --principal-axis-system
```

To rotate the molecule around itself to align with this system do as follows

```
... --rotate-to-principal-axis-system "0,0,1"
```

This rotates the molecule in such a manner that the ellipsoids largest axis is aligned with the z axis. *Note that "0,0,-1" would align anti-parallel.*

Again instead of giving the coordinates explicitly you may also use a vector stored as a geometry object, see section Geometry.

Perform verlet integration

Atoms not only have a position, but each instance also stores velocity and a force vector. These can be used in a velocity verlet integration step. Velocity verlet is an often employed time integration algorithm in molecular dynamics simulations.

```
... --verlet-integration \
    --deltat 0.1 \
    --keep-fixed-CenterOfMass 0
```

This will integrate with a timestep of and correcting forces and velocities such that the sum over all atoms is zero.

Note

Although it may be useful in rare cases to use this Action directly, most of the time it is far more convenient to use the MakroAction molecular-dynamics which incorporates also the force calculation and allows for time integration over more than just a single time step.

Anneal the atomic forces

This will shift the atoms in a such a way as to decrease (or anneal) the forces acting upon them.

Forces may either be already present for the set of atoms by some other way (e.g. from a prior fragmentation calculation) or, as shown here, loaded from an external file. We anneal the forces for one step with a certain initial step width of 0.5 atomic time units and do not create a new timestep for each optimization step.

```
... --force-annealing \
    --forces-file test.forces \
    --deltat 0.5 \
    --steps 1 \
    --output-every-step 0
```

Note

Same as before, although in rare useful, we refer to the MakroAction optimize-structure which incorporates also the force calculation and allows for structure optimization over more than just a single step.

Linear interpolation between configurations

This is similar to verlet integration, only that it performs a linear integration irrespective of the acting atomic forces.

The following call will produce an interpolation between the configurations in time step 0 and time step 1 with 98 intermediate steps, i.e. current step 1 will end up in time step 99. In this case an identity mapping is used to associated atoms in start and end configuration.

```
... --linear-interpolation-of-trajectories \
    --start-step 0 \
    --end-step 1 \
    --interpolation-steps 100 \
    --id-mapping 1
```

Manipulate domain

Here, we elaborate on how to duplicate all the atoms inside the domain, how to scale the coordinate system, how to center the atoms with respect to certain points, how to realign them by given constraints, how to mirror and most importantly how to specify the domain.

Changing the domain

The domain is specified by a symmetric 3x3 matrix. The eigenvalues (diagonal entries in case of a diagonal matrix) give the length of the edges, additional entries specify transformations of the box such that it becomes a more general parallelepiped.

```
... change-box "20,0,20,0,0,20"
```

As the domain matrix is symmetric, six values suffice to fully specify it. We have to give the six components of the lower triangle matrix. Here, we change the box to a cuboid of equal edge length of 20.

Warning

In case of the python interface an upper triangle matrix is given. Hence, the above would read "20,0,0,20,0,20".

Bound atoms inside box

The following applies the current boundary conditions to the atoms. In case of periodic or wrapped boundary conditions the atoms will be periodically translated to be inside the domain again.

```
... --bound-in-box
```

Center atoms inside the domain

This is a combination of changing the box and bounding the atoms inside it.

```
... --center-in-box "20,0,20,0,0,20"
```

Center the atoms at an edge

MoleCuilder can calculate the minimum box (parallel to the cardinal axis) all atoms would fit in and translate all atoms in such a way that the lower, left, front edge of this minimum is at the origin (0,0,0).

```
... --center-edge
```

Extending the boundary by adding an empty boundary

In the same manner as above a minimum box is determined that is subsequently expanded by a boundary of the given additional thickness. This applies to either side, i.e. left and right, top and bottom, front and back.

```
... --add-empty-boundary "5,5,5"
```

This will enlarge the box in such a way that every atom is at least by a distance of 5 away from the boundary of the domain (in the infinity norm).

Scaling the box

You can enlarge the domain by simple scaling factors.

```
... --scale-box "1,1,2.5"
```

Here, the domain is stretched in the z direction by a factor of 2.5. Also, all positions are scaled by the same factor.

Repeating the box

Under periodic boundary conditions often only the minimal periodic cell is stored. E.g. for a crystallic system this minimal cell is all that's needed to completely specify a larger body. If need be, multiple images can be easily added to the current state of the system by repeating the box, i.e. the box along with all contained atoms is copied and placed adjacently.

```
... --repeat-box "1,2,2"
```

This will create a 2x2 grid of the current domain, replicating it along the y and z direction along with all atoms. If the domain contained before a single water molecule, we will now have four of them.

Change the boundary conditions

Various boundary conditions can be applied that affect how certain Actions work, e.g. translate-atoms. We briefly give a list of all possible conditions:

- Wrap

  Coordinates are wrapped to the other side of the domain, i.e. periodic boundary conditions.

- Bounce

  Coordinates are bounced back into the domain, i.e. they are reflected from the domain walls.

- Ignore

  No boundary conditions apply.

The following will set the boundary conditions to periodic.

```
... --set-boundary-conditions "Wrap Wrap Wrap"
```

Note that boundary conditions are not enforced unless explicitly requested, e.g. by the **bound-in-box** action

Filling

Filling a specific part of the domain with one type of molecule, e.g. a water molecule, is the more advanced type of copying of a molecule (see **copy-molecule**) and for this we need several ingredients.

First, we need to specify the part of the domain. This is done via a shape. We have already learned how to create and select shapes. The currently selected shape will serve as the fill-in region.

Then, there are three types of filling: domain, volume, and surface. The domain is filled with a regular grid of fill-in points. A volume and a surface are filled by a set of equidistant points distributed within the volume or on the surface of a selected shape. The latter is closed connected to the respective shape selected. Molecules will then be copied and translated points when they "fit". Note that not only primitive shape can be used for filling in molecules inside their volume or on their surface but also any kind of combined shape.
Note however that not all combinations may already be fully working.

The filler procedure checks each fill-in point whether there is enough space for the set of atoms. To this end, we require a cluster instead of a molecule. A cluster is more general than a molecule as it is not restricted to a connected subgraph with respect to the bond graph. A cluster is just a general agglomeration of atoms combined with a bounding box that contains all of them and serves as its minimal volume. I.e. we need such a cluster. For this a number of atoms have to be specified, the minimum bounding box is generated automatically by which it is checked whether sufficient space is available at the fill-in point.

On top of that, molecules can be selected whose volume is additionally excluded from the filling region.

Fill the domain with molecules

The call to fill the volume of the selected shape with the selected atoms is then as follows,

```
... --fill-regular-grid \
    --mesh-size "5,5,5" \
    --mesh-offset ".5,.5,.5" \
```

```
    --DoRotate 1 \
    --min-distance 1. \
    --random-atom-displacement 0.05 \
    --random-molecule-displacement 0.4 \
    --tesselation-radius 2.5
```

This generates a cardinal grid of 5x5x5 fill-in points within the sphere that are offset such as to lie centered within the sphere, defined by a relative offset per axis in [0,1]. Hence, with an offset of "0" we have the points left-aligned, with "0.5" centered, and with "1" right-aligned. Additionally, each molecule is rotated by a random rotation matrix, each atom is translated randomly by at most 0.05, each molecule's center similarly but at most by 0.4. The selected molecules' volume is obtained by tesselating their surface and excluding every fill-in point whose distance to this surface does not exceed 1. We refer to our comments in Randomizationfor details on changing the randomness and obtaining some extra flexibility thereby.

Fill a shape's volume with molecules

More specifically than filling the whole domain with molecules, maybe except areas where other molecules already are, we also can fill only specific parts by selecting a shape and calling upon the following action:

```
... --fill-volume \
    --counts 12 \
    --min-distance 1. \
    --DoRotate 1 \
    --random-atom-displacement 0.05 \
    --random-molecule-displacement 0.4 \
    --tesselation-radius 2.5
```

The specified option all have the same function as before. Here, we specified to generate 12 points inside the volume of the selected shape.

Fill a shape's surface with molecules

Filling a surface is very similar to filling its volume. Again the number of equidistant points has to be specified. However, randomness is constrained as the molecule has to be aligned with the surface in a specific manner. The idea is to have the molecules point away from the surface in a similar way. This is done by aligning an axis with the surface normal. The alignment axis refers to the largest principal axis of the filler molecule and will be aligned parallel to the surface normal at the fill-in point. This is the same syntax as with **rotate-around-self**.

The call below will fill in 12 points with a minimum distance between the instances of 1 angstroem. We allow for certain random displacements and use the z-axis for aligning the molecules on the surface.

```
... --fill-surface \
    --counts 12 \
    --min-distance 1. \
    --DoRotate 1 \
    --random-atom-displacement 0.05 \
    --random-molecule-displacement 0.4 \
    --Alignment-Axis "0,0,1"
```

Note that instead of giving an explicit axis you may also use a vector stored as a geometry object, see section Geometry.

Suspend in molecule

Add a given molecule in the simulation domain in such a way that the total density is as desired.

```
... --suspend-in-molecule 1.
```

Fill in molecule

This action will be soon be removed.

```
... --fill-molecule
```

Fill void with molecule

This action will be soon be removed.

```
... --fill-void
```

Analysis

If atoms are manipulated and molecules are filled in, it is also a good idea to check on the manner of the filling. This can be done by for example looking at the pair correlation or angular correlation function. This may be useful in checking what is the mean distance between water molecules and how they are aligned with respect to each other.

Pair Correlation

For two given elements Pair correlation checks on the typical distance in which they can be found with respect to one another. E.g. for water one might be interested what is the typical distance for hydrogen and oxygen atoms.

```
... --pair-correlation \
    --elements 1 8 \
    --bin-start 0 \
    --bin-width 0.7 \
    --bin-end 10 \
    --output-file histogram.dat \
    --bin-output-file bins.dat \
    --periodic 0
```

This will compile a histogram for the interval [0,10] in steps of 0.7 and increment a specific bin if the distance of one such pair of a hydrogen and an oxygen atom can be found within its distance interval. These data files can be used for plotting the distribution right away in order to check on the correlation between the elements.

The action may also be called with just a single element in the option *elements*. This will then trigger a pair correlation between all atoms of this element and the set of currently selected atoms.

Dipole Correlation

The dipole correlation is similar to the pair correlation, only that it correlates the orientation of dipoles in the molecular system with one another.

Note that the dipole correlation works on the currently selected molecules, e.g. all water molecules if so selected.

```
... --dipole-correlation \
    --bin-start 0 \
    --bin-width 0.7 \
    --bin-end 10 \
    --output-file histogram.dat \
    --bin-output-file bins.dat \
    --periodic 0
```

Hence, instead of calculating a function of the distance in [0,infinity), it calculates the angular histogram in [0,2pi).

Dipole Angular Correlation

The dipole angular correlation looks at the angles of a dipole over time. It takes the orientation of a certain time step as the zero angle and bins all other orientations found in later time steps relative to it.

Note that in contrast to the dipole correlation the dipole angular correlation works on the molecules determined by a formula. This is because selections do not work over time steps as molecules might change.

```
... --dipole-angular-correlation H2O \
    --bin-start 0 \
    --bin-width 5 \
    --bin-end 360 \
    --output-file histogram.dat \
    --bin-output-file bins.dat \
    --periodic 0 \
    --time-step-zero 0
```

Point Correlation

Point correlation is very similar to pair correlation, only that it correlates not positions of atoms among one another but against a fixed, given point.

```
... --point-correlation \
    --elements 1 8 \
    --position "0,0,0" \
    --bin-start 0 \
    --bin-width 0.7 \
    --bin-end 10 \
    --output-file histogram.dat \
    --bin-output-file bins.dat \
    --periodic 0
```

This would calculate the correlation of all hydrogen and oxygen atoms with respect to the origin.

Naturally, instead of giving explicit coordinates you may also use a vector stored as a geometry object for position, see section Geometry.

Surface Correlation

The surface correlation calculates the distance of a set of atoms with respect to a tesselated surface.

```
... --surface-correlation \
    --elements 1 8 \
    --bin-start 0 \
    --bin-width 0.7 \
    --bin-end 10 \
    --output-file histogram.dat \
    --bin-output-file bins.dat \
    --periodic 0
```

Molecular Volume

This simply calculates the volume that a selected molecule occupies. For this the molecular surface is determined via a tesselation of its surface. Note that this surface is minimal in that respect that each node of the tesselation consists of an atom of the molecule.

```
... --molecular-volume
```

Note

The rolling sphere used in the tesselation algorithm has a default diameter of 10 angström.

Average force acting on a molecule

This sums up all the forces of each atom of a currently selected molecule and returns the average force vector. This should give you the general direction of acceleration of the molecule.

```
... --molecular-volume
```

Fragmentation

Fragmentation refers to a so-called linear-scaling method called "Bond-Order diSSection in an ANOVA-like fashion" (BOSSANOVA), developed by Frederik Heber. In this section we briefly explain what the method does and how the associated actions work.

The central idea behind the BOSSANOVA scheme is to fragment the graph of the molecular system into connected subgraphs of a certain number of vertices, namely a fixed number of atoms. To give an example, loading a ethane atom with the chemical formula C2H6, fragmenting the molecule up to order 1 means creating two fragments, both methane-like from either carbon atom including surrounding hydrogen atoms. Fragmenting up to order 2 would return both the methane fragments and additionally the full ethane molecule as it resembles a fragment of order 2, namely containing two (non-hydrogen) atoms.

The reason for doing this is that usual ab-initio calculations of molecular systems via methods such as Density Functional Theory or Hartree-Fock scale at least as with the number of atoms . In general, this cost is prohibitive for calculating ground state energies and forces (required for molecular dynamics simulations) for larger molecules such as bio proteins. By fragmenting the molecular system and looking at fragments of fixed size, calculating the ground state energy of a number of fragment molecules becomes a linear scaling operation with the number of atoms. In the doctoral thesis of Frederik Heber, it is explained why this is a sensible approach mathematically and shown that it delivers a very good accuracy if electrons (and hence interactions) are in general localized.

Long-range interactions (e.g. Coulomb or van-der-Waals interactions) are artificially truncated, however, with this fragmentation approach. It can be obtained in a perturbation manner by sampling the resulting electronic and nuclei charge density on a grid, summing over all fragments, and solving the associated Poisson equation. Such a calculation is implemented via the solver vmg™ by Julian Iseringhausen that is contained in the ScaFaCoS™ [http://www.scafacos.org/]. This enhancement is implemented via another program package named JobMarket™.

Note that we treat hydrogen special (but can be switched off) as fragments are calculated as closed shell (total spin equals zero). Also, we use hydrogen to saturate any dangling bonds that occur as bonds are cut when fragmenting a molecule (this, too, can be switched off).

Eventually, the goal of fragmentation is to yield an energy and gradients with respect to nuclear positions per atom for each fragment. These forces can be used for molecular dynamics simulations, for structure optimization or for fitting empirical potential functions. The underlying operation consists of three parts. The first part is always the act of splitting up the selected atoms in the molecular system into fragments. The energies of the fragments are calculated in the second part via an external ab-initio solver, where the actual solver used is arbitrary and to some extent up to the user (MPQC ™ is used by default). The second part of the fragmentation is to combine fragment results to energy and gradients for the total molecular system.

Note that the molecular system itself is not touched in any way by this fragmentation.

Fragmenting a molecular system

For the current selection of atoms, all fragments consisting of these (sub)set of atoms are created in the following manner.

```
... --fragment-molecule "BondFragment" \
    --DoCyclesFull 1 \
    --distance 3. \
    --order 3 \
    --grid-level 5 \
    --output-types xyz mpqc
```

We go through each of the options one after the other. During fragmentation some files are created storing state information, i.e. the vertex/atom indices per fragment and so on. These files are used when re-performing the fragmentation on a slightly modified state (e.g. after a time integration step with essentially the same bond graph) for faster operation. These files all need a common prefix, here "BondFragment". Then, we specify that cycles should be treated fully. This compensates for electrons in aromatic rings being delocalized over the ring. If cycles in the graph, originating from aromatic rings, are always calculated fully, i.e. the whole ring becomes a fragment, we partially overcome these issues. This does however not work indefinitely and accuracy of the approximation is limited ( ) in systems with many interconnected aromatic rings, such as graphene. Next, we give a distance cutoff of 3 angstroem used in bond graph creation. Then, we specify the maximum order, i.e. the maximum number of (non-hydrogen) atoms per fragment, here 3. The higher this number the more expensive the calculation becomes (because substantially more fragments are created) but also the more accurate. The grid level refers to the part where long-range Coulomb interactions are calculated. This is done via solving the associated Poisson equation with a multigrid solver -- however, this requires the JobMarket™ package. As input the solver requires the density which is sampled on a cartesian grid whose resolution these parameter defines ( ). And finally, we give the output file formats, i.e. which file formats are used for writing each fragment configuration (prefix is "BondFragment", remember?). Here, we use XYZ (mainly for checking the configurations visually) and MPQC, which is a very robust Hartree-Fock solver. We refer to the discussion of the Parsers on how to change the parameters of the ab-initio calculation. That is to the extent implemented in Molecuilder the created configuration files for the specific solver are manipulated with the options also manipulating the state file written on program exit.

After having written all fragment configuration files, you need to calculate each fragment, grab the resulting energy (and force vectors) and place them into a result file manually. This at least is necessary if you have specified output-types above. If not, the fragments are not written to file but stored internally. Read on.

All created fragments, i.e. their id sets, are stored in an internal structure that associates each atom with all fragments it is contained in. This AtomFragments container structure can be parsed and stored, see

AtomFragments. They are used e.g. for fitting partial charges. There, a selection of atoms is used to fit partial charges to all fragments (and the charge grids obtained from long-range calculations, see FragmentAutomation, and the container is needed to know all fragments.

Note

This structure is cleared by this action and created fragment information is inserted afterwards, i.e. it contains only the associations from the current fragmentation.

Adds selected atoms as one fragment

In cases where you don't want to rely on the above fragmentation# scheme or it is not applicable (e.g., for ions or non-saturated molecules), then you can use the whole following automation mechanism with computing the selected set of atoms as a whole. The following call will treat all atoms per associated molecule as one fragment. These fragments are put either into a job queue (see below on how to continue there),

```
... --add-selected-atoms-as-fragment "BondFragment" \
    --grid-level 5 \
```

or they are stored to files.

```
... --add-selected-atoms-as-fragment "BondFragment" \
    --grid-level 5
    --output-types mpqc
```

Note that the grid-level can be still be specified to also compute long-range interactions in the current domain. Electrostatic interactions within the molecule should generally be accounted for by the employed ab-initio method.

Calculating fragment energies automatically

Another way of doing this is enabled if you have the JobMarket™ package. Initially, the package was required but now it is only recommended as then calculations can be performed in parallel or arbitrarily many cores (and also the long-range calculation are only available then). JobMarket implements a client/server ansatz, i.e. two (or more) independent programs are running (even on another computer but connected via an IP network), namely a server and at least one client. The server receives fragment configurations from MoleCuilder and assigns these to a client who is not busy. The client launches an executable that is specified in the work package he is assigned and gathers after calculation a number of values, likewise specified in the package. The results are gathered together by the server and can be requested from MoleCuilder once they are done. This essentially describes what is happening during the execution of this action.

Stored fragment jobs can also be parsed again, i.e. reversing the effect of having output-types specified in Fragmenting a molecule .

```
... --parse-fragment-jobs \
    --fragment-jobs "BondFragment00.in" "BondFragment01.in" \
    --fragment-path "./" \
    --grid-level 5
```

Here, we have specified two files, namely `BondFragment00.in` and `BondFragment01.in`, to be parsed from the path "./", i.e. the current directory. Also, we have specified to sample the electronic

charge density obtained from the calculated ground state energy solution with a resolution of 5 (see fragment molecule and also below).

This allows for automated and parallel calculation of all fragment energies and forces directly within MoleCuilder. The FragmentationAutomation action takes the fragment configurations from an internal storage wherein they are placed if in FragmentMolecule no output-types have been specified.

```
... --fragment-automation \
    --fragment-executable mpqc \
    --DoLongrange 1 \
    --DoValenceOnly 1 \
    --grid-level 5 \
    --interpolation-degree 3 \
    --near-field-cells 4 \
    --server-address 127.0.0.1 \
    --server-port 1025
```

Again, we go through each of the action's options step by step.

The executable is required if you do not have a patched version of MPQC™ that may directly act as a client to JobMarket's server.

Note

Long-calculations are only possible with a client that knows how to handle VMG jobs. If you encounter failures, then it is most likely that you do not have a suitable client.

In the next line, we have all options related to calculation of long-range interactions. We only sample valence charges on the grid, i.e. not core electrons and the nuclei charges are reduced suitably. This avoids problems with sampling highly localized charges on the grid and is in general recommended. Next, there follow parameters for the multi grid solver, namely the resolution of the grid, see under fragmenting the molecule, the interpolation degree and the number of near field cells. A grid level of 6 is recommended but costly in terms of memory, the other values are at their recommend values.

In the last line, parameters are given on how to access the JobMarket server, namely it address and its port. If the JobMarket package is not available, then these option values cannot be given. Instead the solver is called internally on the same machine and one fragment energy is calculated after the other.

Note

The structure storing the fragment results internally is cleared prior to this action and calculated fragment results is inserted afterwards, i.e. it contains only the calculations from the current run.

Note

All calculated results may be placed in a result file for later parsing, see save fragment results .

Clear fragmentation state

MoleCuilder keeps an internal state about the fragmentation process in order to make calculations more efficient.

This state contains for example the current bond order used at a "site", namely an atom. When the same molecule is fragmented again, then certain results can be reused or simply improved upon.

However, if the molecule has changed, e.g., because the element of an atom was modified which changes saturation and invalidates this caching of results, then this state needs to be reset manually.

This is done by calling the action by

```
... --clear-fragment-state \
```

> **Note**
> This action is at the moment irreversible, i.e. it cannot be undone.

Analyse fragment results

After the energies and force vectors of each fragment have been calculated, they need to be summed up to an approximation for the energy and force vectors of the whole molecular system. This is done by calling this action.

```
... --analyse-fragment-results \
    --fragment-prefix "BondFragment" \
    --store-grids 1
```

The purpose of the prefix should already be known to you. The last option states that the sampled charge densities and the calculated potential from the long-range calculations should be stored with the summed up energies and forces. Note that this makes the resulting files substantially larger (Hundreds of megabyte or even gigabytes as currently the densities are stored on the full cartesian grid). Fragment energies and forces are stored in so-called internal homology containers. These are explained in the next section.

Note that this action sets the force vector if these have been calculated for the fragment. Hence, a verlet integration is possible afterwards.

> **Note**
> If not obtained by fragment automation then fragment results need to be parsed from file, see parse fragment results .

Store a saturated fragment

This will store the currently selected atoms as a fragment where all dangling bonds (by atoms that are connected in the bond graph but have not been selected as well) are saturated with additional hydrogen atoms.

This can work two ways: Either the saturated fragment is written to file. If not "output-types" is given, then the fragment will instead be written to the FragmentQueue. In other words, its energy is calculated when fragment automation is used.

```
... --store-saturated-fragment \
    --DoSaturate 1 \
    --output-types xyz
```

Alternatively, use the following to store in fragment job queue.

```
... --store-saturated-fragment \
    --DoSaturate 1 \
    --grid-level 6
    --max_meshwidth 0.3
```

Parse fragment jobs from files

The fragment jobs that are created by fragment molecule may also be placed in a file for later retrieval. See the details of this action on how to create one file per job.

Later, for parsing these job files, we need to use the parse fragment jobs action as follows:

```
... --parse-fragment-jobs \
    --fragment-jobs "BondFragment00.in" "BondFragment01.in" \
    --fragment-path "./"
```

Here, we give a list of files by the first option and the second option gives an optional path where all these files are stored.

Parse calculated fragment results from file

Fragment results can either be obtained directly from solving the associated electronic structure problem for each of the fragment jobs. Or if that has been done at some earlier stage and results have been written to a file, see save fragment results , then we may also parse these results.

```
... --parse-fragment-results "results.dat"
```

Save calculated fragment results to file

Calculated fragment results may be stored to a single file for later analysis as follows:

```
... --save-fragment-results "results.dat"
```

Note

Beware though that files from long-range calculations may be very large and are stored quite inefficiently at the moment.

Homologies

After a fragmentation procedure has been performed fully, what to do with the results? The forces can be used for time integration and structure optimization but what about the energies? The energy value is basically the function evaluation of the Born-Oppenheimer surface of the molecular system. For molecular dynamics simulations continuous ab-initio calculations to evaluate the Born-Oppenheimer surface, especially the gradient at the current position of the molecular system's configuration, is not feasible. Instead usually empirical potential functions are fitted as to resemble the Born-Oppenheimer surface to a sufficient degree.

One frequently employed method is the many-body expansion of said surface which is basically nothing else than the fragmentation ansatz described above. Potential functions resemble a specific term in this many-body expansion. These are discussed in the next section.

For each of these terms all homologous fragments (i.e. having the same atoms with respect to the present elements and bonded in the same way), differing only in the coordinate of each atom, are just a sampling or a function evaluation of this term of the many-body expansion with respect to varying nuclei coordinates. Hence, it is appropriate to use these function evaluations in a non-linear regression procedure. That is, we want to tune the parameters of the empirical potential function in such a way as to most closely obtain the same function evaluation as the ab-initio calculation did using the same nuclear coordinates. Usually, this is done in a least-square sense, minimising the euclidean norm.

Homologies -- in the sense used here -- are then nothing else but containers for a specific type of fragment of all the different, calculated configurations (i.e. varying nuclear coordinates of the same fragment, i.e. same atoms with identical bonding).

Homologies are created inside MoleCuilder from fragment calculations by using the action

```
... --analyse-fragment-results
```

. This computes the energy of the molecule from the fragments and also the fragment energies. These are put into the internal homology container.

Now, we explain the actions that parse and store those homologies.

```
... --parse-homologies homologies.dat
```

This parses the all homologies contained in the file `homologies.dat` and *appends* them to the homology container. It will also print all found homology graphs and the range of energies calculated for them if the verbosity is suitably set.

```
... --save-homologies homologies.dat
```

Complementary, this stores the current contents of the homology container, *overwriting* the file `homologies.dat`.

```
... --clear-homologies
```

Finally, this clears the current contents of the homology container.

Evaluate Stability

The homologies can be used to quickly get an approximation on the ground state energy of a molecule.

Warning

This does not accurately take the current geometry into account.

In other words, it will produce the equilibrium energy for the given molecule. Using this, there is an action that evaluates the stability of a molecule by removing each bond, saturating the ends and comparing the ground state energy of the two products to the educt plus the ground state energy of as many hydrogen molecules to match the cut bond's degree.

```
... --evaluate-stability output.csv
```

This writes a table of educt and product names along with their summed energies. The comparison between educt and product energy in each row yields whether this molecule is overall stable.

Note

Be aware that this calculation is based on saturation, i.e. hydrogens are added to saturate dangling bonds. These additional atoms need to be compensated and for this reason there is an additional educt, a number of hydrogen molecules.

AtomFragments

Similarly, to Homologies also the associations between atoms and the respective fragments they take part in can be stored to a file and parse again at a later time.

```
... --parse-atom-fragments atomfragments.dat
```

This parses the all atom fragment associattions contained in the file `atomfragments.dat` and *appends* them to the atom fragments associations container.

```
... --save-atom-fragments atomfragments.dat
```

Complementary, this stores the current contents of the atom fragments associations container, *overwriting* the file `atomfragments.dat`.

Potentials

In much the same manner as we asked before: what are homology files or containers good for? However, taking into account what we have just explained, it should be clear: We fit potential function to these function evaluations of terms of the many-body expansion of the Born-Oppenheimer surface of the full system.

Adding potentials

First of all, potentials can be added using

```
... --add-potential Morse \
    --potential-charges 8 1
```

This will add a "Morse" potential between the elements carbon(8) and hydrogen(1) (the order
<emph>not</emph>
always irrelevant. Here, for this pair potential it is).

Removing potentials

And similarly, potentials can be removed using

```
... --remove-potential Morse \
    --potential-charges 8 1
```

This will remove a "Morse" potential between the elements carbon(8) and hydrogen(1).

Note

Typically, however, you do not want to add potentials this way. They lack the parametrization that make them any useful. In the following, we present more actions for fitting the potential to calculcated energies and forces and thus obtaining such a useful parametrization.

Fitting empirical potentials

Empirical potentials are function that represent a certain aspect of binding forces in molecular dynamics, e.g. a bond potential represents the force between two atoms arising because of a binding orbital in between that may be (in some approximartion) represented by a Hooke's spring law. In a more abstract view, an empirical potential consists of the following: a binding model representable as a graph consisting of nodes and edges, a function that takes the distance between nodes and the set of edges (representing bonds in the binding model) and last but not least a set of parameters that represent the respective strength of the bonds. For example, a torsion potential has a binding model containing four nodes and three edges connecting node 1 and 2, 2 and 3, and 3 and 4. The function requires the six interatomic distances. And the parameters are coefficients in the functions that need to be evaluated to obtain the resulting force vector.

In this manner, empirical potentials are implemented in MoleCuilder. They are not just a function but always an additional binding model that allows to associate atoms with a specific node in the model and thereby to assicate it with a particular interatomic distance. And said model determines in what order the elements have to be given.

Let's take a look at an exemplary call to the fit potential action.

```
... --fit-potential \
    --fragment-charges 8 1 1 \
    --potential-charges 8 1 \
    --potential-type morse \
    --take-best-of 5
```

Again, we look at each option in turn. The first is the charges or elements specifying the set of homologous fragments that we want to look at. Here, obviously we are interested in water molecules, consisting of a single oxygen (8) and two hydrogen atoms (1). Next, we specify the chemical element type of the potential, here a potential between oxygen (8) and hydrogen (1). We give the type of the potential as morse, which requires a single distance or two nuclear coordinates and the distance taken between the two. Finally, we state that the non-linear regression should be done with five random starting positions, i.e. five individual minimizations, and the set of parameters with the smallest L2 norm wins.

Note

Due to translational and rotational degrees of freedom for fragments smaller than 7 atoms, it is appropriate to look at the pair-wise distances and not at the absolute coordinates. In the case of the water molecule as a the fragment whose energy we want to represent by a empirical potential, there are 3 atoms and therefore 3 unique distances between any pair of atoms. From this set of distances MoleCuilder needs to pick any subset that matches with the ones required by the binding model. In our case of the Morse potential, we need two atoms, oxygen and hydrogen, i.e. a single distance. If we had given a harmonic angular potential and the then required three charges/elements, "1 8 1", i.e. oxygen and two hydrogens, we would have obtained three distances. The order of the elements, i.e. "8 1 1" would match a different angular interaction in the same fragment, depends on the binding model of the potential. In the case of the harmonic angle, the second element in the list is the central atom in the angle, while the first and third atom define either arm of the angle.Naturally, for the Morse potential the order does not matter as each distance is symmetric.

MoleCuilder always adds a so-called constant potential to the fit containing only a single parameter, the energy offset. This offset compensates for the interaction energy associated with a fragment of order 1, e.g. a single hydrogen atom. Essentially, this captures the atomic energy that is not associated to any binding interactions.

Note that by choosing "set-max-iterations" and "take-best-of" one can force the optimization to try either a single set of random initial parameters very thoroughly or many different sets just for a few iterations. Or any in between.

Fitting many empirical potentials simultaneously

Another way is using a file containing a specific set of potential functions, possibly even with initial values.

The .potentials file needs to be loaded beforehand by using the parse potentials action. After the fit it needs to be saved using save potentials to have it contain the updated values.

```
... --fit-compound-potential \
    --fragment-charges 8 1 1 \
    --set-threshold 1e-3 \
    --training-file test.dat
    --error-file error.dat
```

Now, all empirical potential functions are summed up into a so-called compound potential over the combined set of parameters. These are now fitted simultaneously. For example, let's say the potential file `water.potentials` contains a harmonic bond potential between oxygen and hydrogen and another angular potential for the angle between hydrogen, oxygen, and hydrogen atom. Then, we would fit a function consisting of the sum of the two potentials functions in order to approximate the energy of a single water molecule (actually, it's the sum of three potentials. As mentioned before, a constant potential is always added to compensate non-bonding energies, i.e. not depending on interatomic distances). Here, the threshold criterion takes the place of the **take-best-of** option. Here, the minimization is reiterated so often on random (but to some extent chosen from a sensible range) starting parameters until the final L2 error is below 1e-3. Also, all data used for training, i.e. the tuples consisting of the fragments nuclei coordinates and the associated energy value are written to the file `test.dat`. This allows for graphical representation or other way of analysis, e.g. for a Morse potential between oxygen and hydrogen the bonding energy can be plotted as a one-dimensional function and compared to the "point cloud" of sample points from the fragment term of Born-Oppenheimer surface. It is this point cloud, i.e. the training data, that is written to the file `test.dat`. Moreover, the error per fragment is written to the file `error.dat`

Note that you can combine the two ways, i.e. start with a fit-potential call but give an empty potential file. The resulting parameters are stored in it. Fit other potentials and give different file names for each in turn. Eventually, you have to combine the file in a text editor at the moment. And perform a fit-compound-potential with this file.

Here, also "set-max-iterations" and "take-best-of" can be used to mix thorough or shallow search from random initial parameter sets.

Parsing an empirical potentials file

Taking part in the compound potential is every potential function whose signature matches with the designated fragment-charges and who is currently known to MoleCuilder.

More potentials can be registered (**fit-potential** will also register the potential it fits) by parsing them from a file.

```
... --parse-potentials water.potentials
```

Note

Currently, only TREMOLO™potential files are understood and can be parsed.

Generate empirical potentials

The actions to fit potentials take either a single potential name or a potentials file. In order to create all potentials that would match with its particle types the following action can be called:

```
... --generate-potentials \
--fragment-charges 8 1 1
```

Here, it would create all potentials for a fragment specifying a water molecule: harmonic_bond for O and H, morse for O and H, harmonic_angle for H, O, and H (and not for O, H, H, or other combinations as these subgraphs do not exists for the molecule H-O-H), and tersoff for the types O and H.

```
... --generate-potentials \
--fragment-charges 8 1 1 \
              --potential-list constant harmonic_bond harmonic_angle
```

This would generate the potentials taken from the restricted set of listed potential names. If the give list is empty (i.e. the parameter is unset), then by default all available potentials are used.

Saving an empirical potentials file

The opposite to **parse-potentials** is to save all currently registered potential functions to the given file along with the currently fitted parameters

```
... --save-potentials water.potentials
```

Note

Again, only the TREMOLO™potential format is understood currently and is written.

Fitting partial particle charges

The above empirical potential just model the short-range behavior in the molecular fragment, namely the (covalently) bonded interaction. In order to model the Coulomb long-range interaction as well without solving for the electronic ground state in each time step, partial charges are used that capture to some degree the created dipoles due to charge transfer from one atom to another when bonded. These are called partial charges because they combine both nuclei and electronic charges to yield an in general fractional charge.

Note that so far the placement of partial charges is restricted to the position of nuclei in the molecular system. There are more complex ways of placing partial charges, e.g. as employed in higher TIP water molecules, that also use anti-bonding potentials. This is so far not implemented.

To allow least-squares regression of these partial charges, we need the results of long-range calculations and the **store-grids** option (see above under Fragmentation) must have been given.

Furthermore, we require associations between selected atoms and the fragments, residing in the Homology container. These are contained in the AtomFragments association container, that can also be parsed and stored.

With these sampled charge density and Coulomb potential stored in the homology containers, we call this action as follows.

```
... --fit-partial-charges \
    --potential-file water.particles \
    --radius 1.5
```

Assume that a water molecule has been selected previously. Then all homologous fragments that contain any of the water molecules are used as "key" to request all configurations of this type from the homologies container. For each of the atoms then an average partial charge is computed by fitting their respective Coulomb potential to the obtained from the fragment calculations. Resulting values are stored in `water.particles`. The radius is used to mask a certain region directly around the nuclei from the fit procedure. As here the charges of the core electrons and the nuclei itself dominate, we however are only interested in a good approximation to the long-range potential, this mask radius allows to give the range of the excluded zone.

Parsing a particles file

Empirical Potential contain parameters for function that model interactions between two or more specific particles. However, interactions can also be more general, such as a Coulomb potential, that in-

teracts with any other particle with non-zero charge, or Lennard-Jones potential that interacts with any other particle close enough. Parameters for these particles are encoded in the internal Particle class and are parsed from and stored to a special particles file. This parse particle parameters function loads the parameters from a file, instantiates them in a new particle, and registers them such that they are known within molecuilder.

More particles can be registered (**fit-partial-charges** will also register the particles it fits) by parsing them from a file.

```
... --parse-particle-parameters water.particles
```

Note

Currently, only TREMOLO™particles files are understood and can be parsed.

Saving a particles file

The opposite to **parse-particle-parameters** is to save all currently registered particle and their parameters to the given file.

```
... --save-particle-parameters water.particles
```

Note

Again, only the TREMOLO™particles format is understood currently and is written.

Dynamics

For fitting potentials or charges we need many homologous but different fragments, i.e. atoms with slightly different positions. How can we generate these?

One possibility is to use molecular dynamics. With the aforementioned fragmentation scheme we can quickly calculate not only energies but also forces if the chosen solver, such as MPQC ™ [http://www.mpqc.org/], supports it. Integrating these forces discretely over time gives insight into vibrational features of a molecular system close to the equilibrium and allows to generate those positions for fitting potentials that describe these vibrations.

Molecular dynamics

The molecular dynamics action is a so-called macro Action, i.e. it combines several other Actions into one, namely:

- **--verlet-integration**
- **--output**
- **--clear-fragment-results**
- **--destroy-adjacency**
- **--create-adjacency**
- **--update-molecules**
- **--fragment-molecule**

- **--fragment-automation**
- **--analyse-fragment-results**

The following command will perform a molecular dynamics simulation for 100 time steps, each time step continuing over **deltat** equal to 0.5 atomic time units, i.e. 1.2e-17 s. Some of the other options listed below, such as **order**, **distance**, or **fragment-executable**, will seem familiar to you if you have read in this guide about the fragmentation actions. Look at the list above to see that they are a part of the makro action that performs molecular dynamics simulations. Below we will not keep the bond graph, i.e bonds and molecules may change over the simulation and hence also the created fragments per time step. Furthermore, the forces are corrected such that the force add up to zero.

```
... --molecular-dynamics \
    --steps 100 \
    --keep-bondgraph 0 \
    --order 3 \
    --distance 3. \
    --deltat 0.5 \
    --keep-fixed-CenterOfMass 1 \
    --fragment-executable mpqc \
```

Keeping the bond graph is useful when simulating close to the equilibrium and no bond breaking or forming should occur. Note that in general the BOSSANOVA fragmentation scheme is discontinuous with respect to the formation of bonds in the energy. This discontinuity arrises because of the threshold criterion used for detecting the bond graph. After each simulation step the bond graph is recreated (if **keep-bondgraph** is switched off) to accomodate for any structural changes. If bonds are added because two atoms are suddenly within the required distance which before was not the case, additional fragments are generated, calculated, and added to the approximation of the whole system. As the addition of these contributions is sudden -- in general, they will already be non-zero when the bonds are detected -- a slight jump in the total energy of the system can be expected.

To sum it up, the use of the BOSSANOVA scheme in bond breaking/forming scenarios is in general not recommended. That's why keeping the bond graph is always sensible. However, if potential energies are too far away from equilibrium the simulation may still produce unphysical results.

Structure optimization

Structure optimization is also a macro Action, it basically combines the same Actions as **molecular-dynamics** does. However, it uses the **force-annealing** action instead of **verlet-integration**.
Because it is a MacroAction we cannot use any more elaborate line search method during the optimization. In essence, we minimize the energy function that depends on the nuclei coordinates. The function is non-linear and non-convex. The best choice, given that we have gradient information, would be the Conjugate Gradient method (such as Fletcher- Reeves which work well also for non-linear functions). However, these methods perform a line search along the gradient direction which we cannot as our Actions do not contain any internal information (apart from the state for Undo/Redo). Therefore, we are restricted to the method of Barzilai-Borwein where no line-search is needed but that also works well for high-dimensional minimization problems.

The command below performs a structure optimization of the currently selected atoms (may also be a subset) for up to 100 time steps, where each time step is again 0.5 atomic time units. The time step here serves as the initial step width for annealing.

```
... --optimize-structure \
    --keep-bondgraph 1 \
    --output-every-step 1 \
    --steps 100 \
    --order 3 \
    --distance 3. \
```

```
--deltat 0.5 \
--keep-fixed-CenterOfMass 1 \
--fragment-executable mpqc
```

Note that **output-every-step** will allow you to watch the optimization as each step is placed into a distinct time step. Otherwise only two time steps would be created: the initial and the final one containing the optimized structure.

Because of the use of Barzilai-Borwein for computing the stepwidth we do not need any 'deltat' parameters. If the computed step width is zero, we use a default step width of 1.

... using the bond graph

A more efficient optimization, especially for larger molecules is obtained, if the bond graph is taken into account. To this end, the structure optimization can be called as follows:

```
... --optimize-structure \
      --keep-bondgraph 1 \
      --output-every-step 1 \
      --steps 100 \
      --order 3 \
      --distance 3. \
      --deltat 0.5 \
      --keep-fixed-CenterOfMass 1 \
      --fragment-executable mpqc \
      --use-bondgraph 1 \
      --damping-factor 0.5 \
      --max-distance 8
```

Note that two additional arguments **use-bondgraph**, **damping-factor**and **max-distance**. The first will switch on using the bond graph while the latter two are parameters controlling its behavior.

Let us briefly sketch the central idea of using the bond graph: If a specific atom has a non-zero gradient, then this gradient will cause the atom to be shifted into its negative direction. The gradient however is a sum of forces acting on this atom from all other atoms. The gradient therefore states the net effect on the toal energy if the atom (and only the atom) would be moved according to its negative direction. In other words, the force on one side of the atom are stronger than those on the other side, where the sides are defined by the plane with the gradient as its normal vector.

The idea now is to not only move the atom itself but also all atoms in the direction of the negative gradient by a fraction as well. The underlying notion is that if there is a misplacement of the said atom under consideration, then this misplacement will faster spread out to the edges of the molecule and dissipate there. If we move only the atom, this will cause a lot of oscillations that will take a while to settle. One might think of it like the smoothing step of a multigrid solver. Essentially, we consider the gradient as a local error that is smoothed out by powers of **damping-factor** the further out we go in the bond graph from the respective atom till **max-distance**. Note that the distance is here to be understood in the sense of a graph, i.e. stepping from one atom to a bond neighbor in the bond graph is a distance of 1.

Note

The truncated power series of the **max-distance** should sum to 1, i.e. values between 0.5 and 1 work well.

Note

Hydrogens due to their typically much ligher mass compared to other nuclei are excluded from this procedure. They feel the dampened gradients of others but their gradient acts only on themselves

Note

Barzilai-Borwein step width sometimes tends to overshoot as it is a approximation to the secant and assumes linearity. To counter this, we cap the step width at 0.2 angstroem.

Step forward and backward through world time

Some MacroActions are applied for a number of steps and need to increment the current world time, e.g. molecule dynamics or structure optimization. To this end, we may call upon

```
... --step-world-time 1
```

Note that the argument gives the number of steps to step forward and may be any integer. Hence, we may also step backwards.

Set the world's time step

In order to inspect or manipulate atoms and molecules at a certain time step, the World's time has to be set with the following Action.

This will set the World's time to the fifth step (counting starts at zero).

```
... --set-world-time 4
```

Note that each atom has its own tracjectory storage and manages it in a clever way. That's why subsets of atoms may be time-integrated, while the other atoms will not get taken over on the time axis. They simply remain frozen during the integration.

Save the temperature information

For each present time step the temperature (i.e. the average velocity per atom multiplied with its mass), momentum and force will be stored to a file.

```
 ... --save-energies temperature.dat
```

That is *temperature.dat* contains two columns: the first contains the time step and the second column contains the temperature of the system in atomic units.

Tesselations

A tesselation is a set of triangles that form a closed surface. They are used to obtain molecular surfaces (and volumes) by rolling a virtual sphere of a certain radii on a molecule such that it always rests on at least three atoms. From such a resting position the sphere is rolled over all of its three sides until it rests again. This is continued until the closed surface of connected triangles is created.

Note

Tesselations are used internally by the graphical interface in order to show molecules by their surface. This is in general faster than displaying them as a ball-stick model consisting of spheres and cylinders.

Non-convex envelope

This will create a non-convex envelope for a molecule and store it to a file for viewing with external programs.

```
... --nonconvex-envelope 6. \
    --nonconvex-file nonconvex.dat
```

This tesselation file can be conveniently viewed with TecPlot™ or with one of the Tcl script in the *util* folder with VMD™. Also, still pictures can be produced with Raster3D ™.

Note

The required file header.r3d can be found in a subfolder of the util folder.

Convex envelope

This will create a convex envelope for a molecule and give the volumes of both the non-convex and the convex envelope. This is good for measuring the space a molecule takes up, e.g. when filling a domain and taking care of correct densities.

```
... --convex-envelope 6. \
    --convex-file convex.dat
```

This tesselation file can be likewise viewed with TecPlot™ or with one of the Tcl script in the util folder with VMD™.

Various commands

Here, we gather all commands that do not fit into one of above categories for completeness.

Changing verbosity

The verbosity level is the amount of stuff printed to screen. This information will in general help you to understand when something does not work. Mind the *ERROR* and *WARNING* messages in any case.

This command below sets the verbosity from default of 2 to 4,

```
... --verbose 4
```

or shorter,

```
... -v 4
```

Dry runs

A "dry run" refers to a test run where commands are not actually executed. You may bracket a certain set of actions by putting **--dry-run** before and **--no-dry-run** afterwards. Then, all actions in between will be looked at but not executed, i.e. they make it to the history but nothing is changed in the World.

As an example, the following listing contains the adding of a hydrogen atom at position (5,5,5) inside the aforementioned dry run statements. Hence, no hydrogen atom is added but the **add-atom** action is stored in the history and will make it to a stored session.

```
... --dry-run \
    --add-atom 1 --domain-position "5,5,5"
    --no-dry-run
```

This is useful for converting shell commands into python scripts. Commands are not executed but all are eventually found in the written pyrthon file.

Loading an element database

Element databases contain information on valency, van der Waals-radii and other information for each element.

This loads all element database from the current folder (in a unix environment):

```
... --element-db ./
```

Giving the version of the program

This prints the version information of the code, especially important when you request the fixing of bugs or implementation of features.

```
... --version
```

Giving warranty information

As follows warranty information is given,

```
... --warranty
```

Giving redistribution information

This gives information on the license and how to redistribute the program and its source code

```
... --help-redistribute
```

Sessions

A session refers to the queue of actions you have executed. Together with the initial configuration (and all files required for actions in the queue) this may be seen as a clever way of storing the state and history of a molecular system manipulation session. When proceeding in a try&error fashion to construct a certain system, it is a good idea, to store the session at the point where your attempts start to deviate from one another.

Note only that but stored session used in the graphical interface may safe a lot of repetitive pointing and clicking. On top of that if the python session file is called *molecuilder.py* and resides in the folder you start MoleCuilder from, then it will be executed automatically and even before creating the graphical interface (i.e. it is also faster).

Storing a session

Storing sessions is simple,

```
... --store-session "session.py" \
    --session-type python
```

Here, the session type is given as python (the other option is " cli" for storing in the manner of the command-line interface, i.e. just as our example code snippets throughout this guide). The written python script `session.py` can even be used with the python interface described below, i.e. it is a full python script (that however requires the so-called *pyMoleCuilder* module).

Note

The python session will store a file with python commands using named arguments. As some actions have quite a number of arguments, all of them end up in a single function call. This makes it very hazard-prone to mix them up. Therefore, it is *strongly* recommended to always use named arguments in python scripts when employing MoleCuilder commands.

When using named arguments for MoleCuilder commands in Python scripts, remember that dashes ("-") in argument names have been converted to underscores ("_") as the former characters are illegal. Apart from that the argument names resemble exactly the token names as encountered on the command line, e.g.

```
pyMoleCuilder.CommandVerbose(verbose="1")
...
```

where the keyword argument "verbose" is for the Action that changes MoleCuilder's output verbosity. Also, you see that arguments are always given as string.

Loading a session

Loading a session only works for python scripts, i.e. only for session type python and not for type cli. This actually blurs the line between the command-line interface and the python interface a bit. Again, MoleCuilder automatically executes a script called `molecuilder.py` if such a file is contained in the current directory.

```
... --load-session "session.py"
```

This will execute every action with its options contained in the script `session.py`.

Various specific commands

In this (final) section of the action description we list a number Actions that are very specific to some purposes (or other programs).

Saving exttypes of a set of atoms

This saves the atomic ids of all currently selected atoms in a TREMOLO ™ [http://www.tremolo-x.com/] exttypes file with the given name.

```
... --save-selected-atoms-as-exttypes \
    --filename test.exttypes
```

Setting parser specific parameters

You can tweak the parameters stored in files associated to specific ab initio programs to some extent. For example, MPQC™ stores various parameters modifying the specific ab-initio calculation performed, e.g. the basis set used, the level of theory, whether gradients are calculated. For MPQC ™ [http://www.mpqc.org/] and Psi4 ™ [http://www.psicode.org/] this can be modified as follows.

```
... --set-parser-parameters mpqc \
    --parser-parameters "theory=CLHF;basis=6-31*G;"
```

This sets the ab-initio theory to closed-shell Hartree-Fock and the basis set to 6-31*G. Note the specific "key=value;" system. That is a key such as "theory" is followed by an equivalent sign and by a value, here "CLHF" for closed-shell Hartree-Fock, and finally a semicolon to separate the key-value pairs. Please avoid any unnecessary white spaces. Note that the implementation is probably not complete, hence do check the MPQC™ manual on further supported values that must be added by hand. We list below the currently implementd list of keys and their values.

- Hessian - yes/no
- savestate - yes/no
- do_gradient - yes/no
- maxiter - positive integer value
- memory - positive integer value

- stdapprox
  - A'
- nfzc - positive integer value
- basis - any basis listed in data folder, e.g.
  - 3-21G
  - 6-31+G*
  - ...
- aux_basis - same as under basis
- integration
  - IntegralCints
- theory
  - CLHF
  - CLKS
  - MBPT2
  - MBPT2_R12
- jobtype
  - Default
  - Optimization

Tremolo specific options and potential files

TREMOLO™'s configuration files start with a specific line telling the amount of information that is contained in the file. This line can be modified, e.g. to enforce storing of velocities and forces as well as the atoms positions and element.

```
... --set-tremolo-atomdata "id element u=3 v=3 F=3" \
    --reset 1
```

This will not append but reset the old line and fill it with the given string.

Find a list of all keys that are also used internally in MoleCuilder below:

- id number - Id
- chemical element - type
- position - x=3
- velocity - u=3
- force - F=3
- enumerating bonded atom ids - neighbors=4

- name of atom - name
- charge - Charge

Furthermore, find the rest of all available keys:

- stress tensor - stress
- enumerating bonded atom ids for improper potential - imprData
- unknown - GroupMeasureTypeNo
- group association for external forces - exttype
- residual name of atom (see PDB) - resName
- chain Id (see PDB) - chainId
- residual sequence (see PDB) - resSeq
- occupancy of electron orbitals - occupancy
- temperature factor - tempFactor
- segment id (see PDB) - segID
- different charge - charge
- unknown - GrpTypeNo
- enumerating bonded atom ids for torsion potential - torsion

One further specific action is required when loading certain TREMOLO™ configuration files. These contain element notations that refer to parameterized names used in empirical potentials and molecular dynamics simulations and not the usual chemical symbols, such as H or O. We may load an auxiliary file that gives the required conversion from OH1 to H, which is the so-called potentials file.

```
... --parse-tremolo-potentials water.potentials
```

This parses the lookup table from the file `water.potentials` and it can be used in following load actions.

Note that this is the same format as written by the fitting actions. However, apart from this the Actions are not related, i.e. **parse-potentials** will not also load these element descriptions. If this is desired, the above Action has to be used.

Text menu

We now discuss how to use the text menu interface.

The text menu is very much the interface counterpart to the command-line interface. However, both work in a terminal session.

In the text menu, actions can be selected from hierarchical lists. Note that the menus for the graphical interface are organized in the exactly same way. After an action has been chosen, the option values have to be entered one after the other. After the last option value has been given, the action is executed and the result printed to the screen.

With regards to the other functionality, it is very much the same as the command-line interface above. It differs by being interactive, i.e. when using an external viewer on the state file and saving it after an

Action has been performed, output can be checked. However, you may work in this way also directly by using the graphical interface.

Graphical user interface

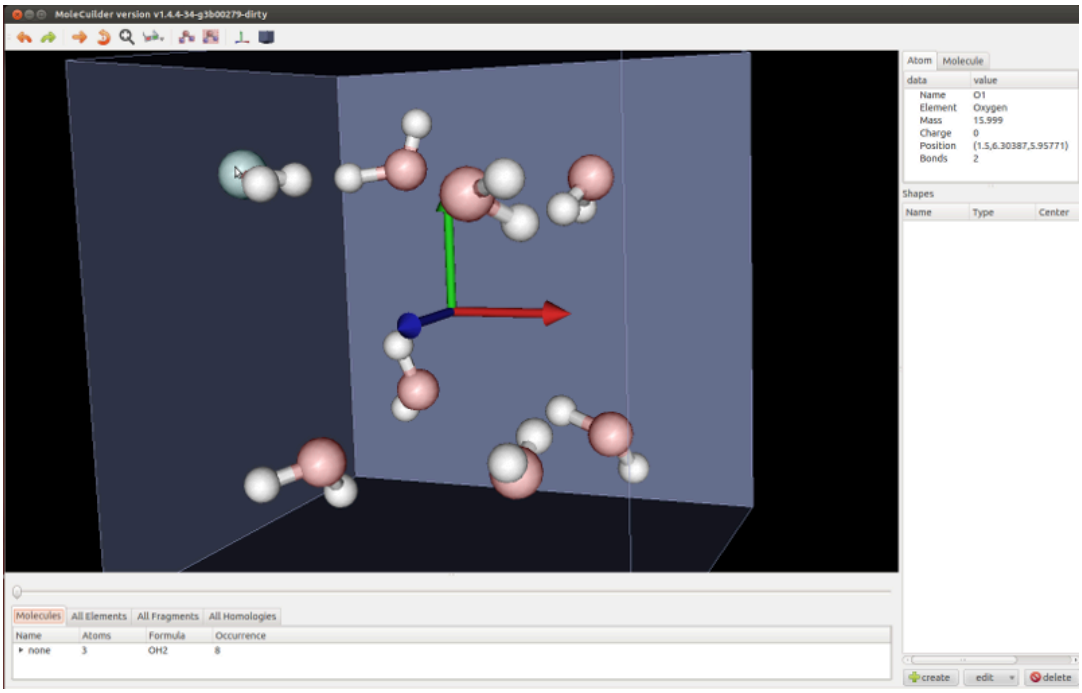
The main point of the GUI is that it renders the atoms and molecules visually. These are represented by the common stick-and-ball-model by default. For faster rendering, molecules can also be visualized by the a non-convex envelope, see Tesselations. Single or multiple atoms and molecules can easily be accessed, activated, and manipulated via tables. Changes made in the tables cause immediate update of the visual representation. Under the hood each of these manipulations is nothing but the call to an action, hence is fully undo- and redoable.

This interface is most helpful in designing more advanced structures that are conceptually difficult to imagine without visual aid. Results can be inspected directly and, if unsuitable, can be undone and reperformed with tweaked parameters. At the end, a file containing the session may be stored and this script can then be used to construct various derived or slightly modified structures.

Basic view

Let us first give an impression of the basic view of the GUI after a molecule has been loaded.

Figure 3.1. Screenshot of the basic view of the GUI after loading a file with eight water molecules.



3D view

In the above figure, you see the stick-and-ball representation of the water molecules, the "dreibein" giving the positive axis directions and the slightly translucent cuboid of the domain on a black background.

Mouse Controls

The viewport can be most easily changed with the mouse: Clicking and holding the left mouse button while moving the mouse will rotate the viewport.

If you hold the Shift key at the same time, the mouse movements will no cause a rotation but a translation in the vertical and horizontal directions, orthogonal to the current view axis.

Finally, the mouse wheel allows to zoom in and out.

There is one more control: right-clicking with the mouse anywhere in the domain will open a context menu specific to atoms, allowing you to add atoms, saturate them, ...

Key Controls

The 3D view can also be controlled through the following keys:

Table 3.1. Keymap

Key	Action
Cursor Up	Moves the view upwards
Cursor Down	Moves the view downwards
Cursor Right	Moves the view to the right
Cursor Left	Moves the view to the left
Page Up	Moves the view further away
Page Down	Moves the view closer
Shift	Holding shift key switches mouse movements from rotation to translation mode

Information Tabs

Beneath this 3D view that you can rotate at will with your mouse and zoom in and out with your scroll wheel, you find to the right a part containing two tabs named Atom and Molecule. Look at where the mouse pointer is. It has colored the atom underneath in cyan (although it's also an oxygen atom and should be colored in rose as the rest). You can inspect its properties in the tab Atom: Name, element, mass, charge, position and number of bonds. If you switch to the Molecule tab, you would see the properties of the water molecule this specific atom belongs to.

Shape section

Beneath these information tabs you find the shape sections. There you find a list of all currently created shapes and you can select them by clicking on them, which lets their surface appear in the 3D view, and manipulate them via the buttons beneath this list. Note that the calculation of the shape's surface may take up to a few seconds, so don't get itchy right away when nothing seems to happen for a moment.

Timeline

Directly below the 3D view there is a long slider. If a loaded file has multiple time step entries, this slider allows you to smoothly select one time frame after another. Sliding it with the mouse from left to right will reveal the animation that is hidden behind the distinct snapshots stored in the configuration file.

Selection tables

Underneath the time line there is another place for tabs.

- Molecules
- All Elements
- All Fragments

- All Homologies
- All Geometries
- Logs
- Errors

The first is on molecules, listing all present molecules of the molecular system in a tree view. If you click on a specific molecule, the one will get selected or unselected depending on its current selection state (see below for details on this with respect to the GUI). Also, if you tick the box the visibility of the specific molecules is changed: it switches between being displayed as either ball-and-stick, for manipulating its individual atoms, or as a molecular surface for (un)selecting it as a whole.

The next tab enumerates all elements known to MoleCuilder where the ones are grayed out that are not present in the molecular system. Clicking on a present element will select all atoms of this specific element. A subsequent click unselects again.

Subsequently follow two tabs on enumerating the fragments and their fragment energies if calculated and the homologies along with graphical depiction (via QWT), again if present.

After that, we have a tab listing all geometry objects. These are vectors you may store via one of the Actions. If you hover over a vector, its length is shown. If you have selected one vector and hover over another one, then the angle between the two is shown.

Finally, there are two tabs showing log messages of actions in the first tab and general information on what is currently done. Errors and warnings are listed in the second tab.

Selections

Selections work generally always by calling a selection action from the pull-down menu and filling it with required parameters.

With the GUI it may also be accessed directly: The row of icons above the 3D view has two icons depicting the selection of individual atoms or molecules. If either of them is selected, clicking with the left mouse button on an atom will either (un)select the atom or its associated molecule. Multiple atoms can be selected in this manner.

Also, the selection tabs may be used by clicking on the name of a molecule as stated above or at an element.

Similarly, if shapes are present in the shape section, clicking them will select them and also cause a translucent visualization to appear in the 3D view. Note that this visualization is quite costly right now and not suited to complex shapes. (This is in contrast to the molecular surfaces which are actually cheaper than the ball-and-stick presentation).

Dialogs

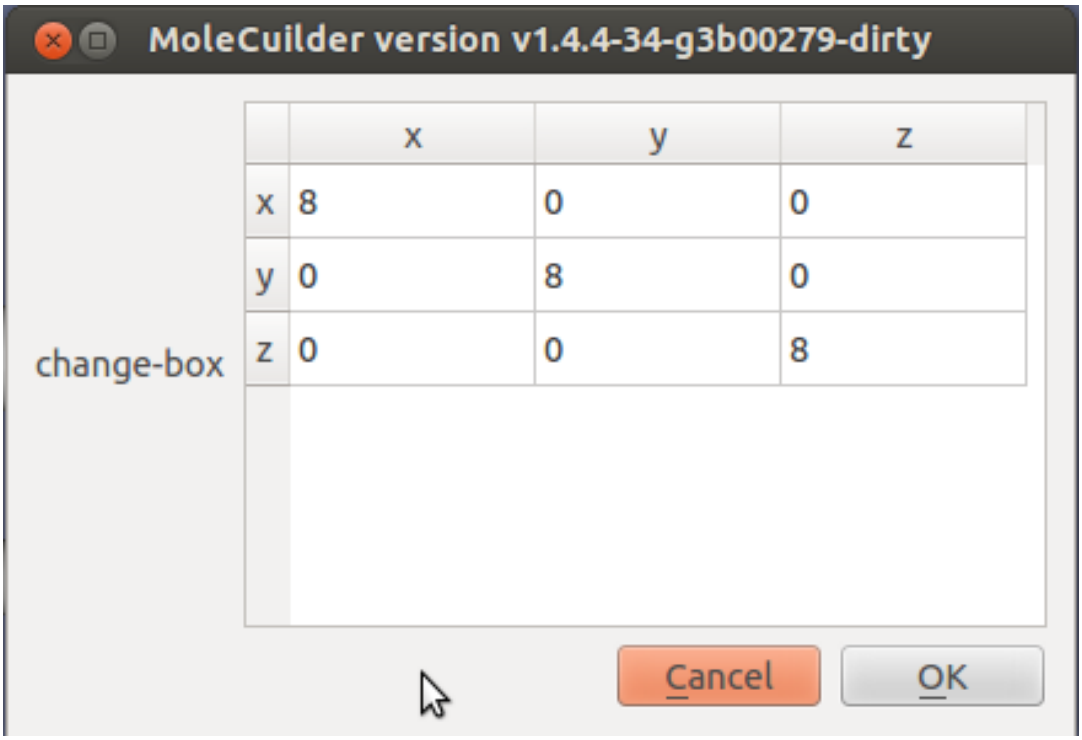
Most essential to the GUI are dialogs. Many action calls forth such a dialog. A dialog consists of a list of queries for a particular option value, one below the other. As each option value has a specific type, we briefly go into the details of how these queries look like.

Note

Each dialog's okay button is grayed out until all entered option values are valid.

Domain query

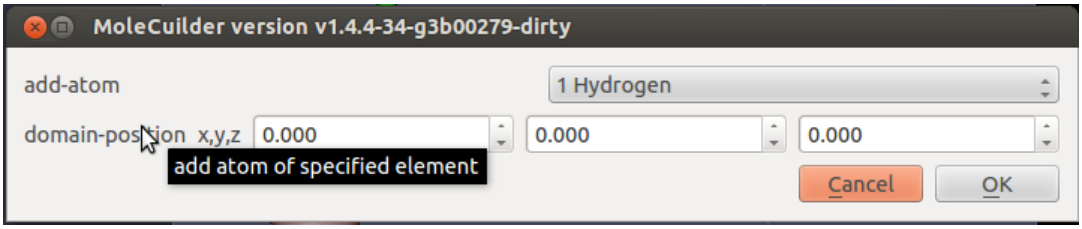
Figure 3.2. Screenshot of a dialog showing a domain query



In the domain query a 3x3 symmetric matrix has to be entered. In the above screenshots you notice that the only non-zero entries are on the main diagonal. Here, we have simply specified a cube of edge length 8. The okay button will be grayed out if the matrix is either singular or not symmetric.

Element query

Figure 3.3. Screenshot the add atom action containing an element query

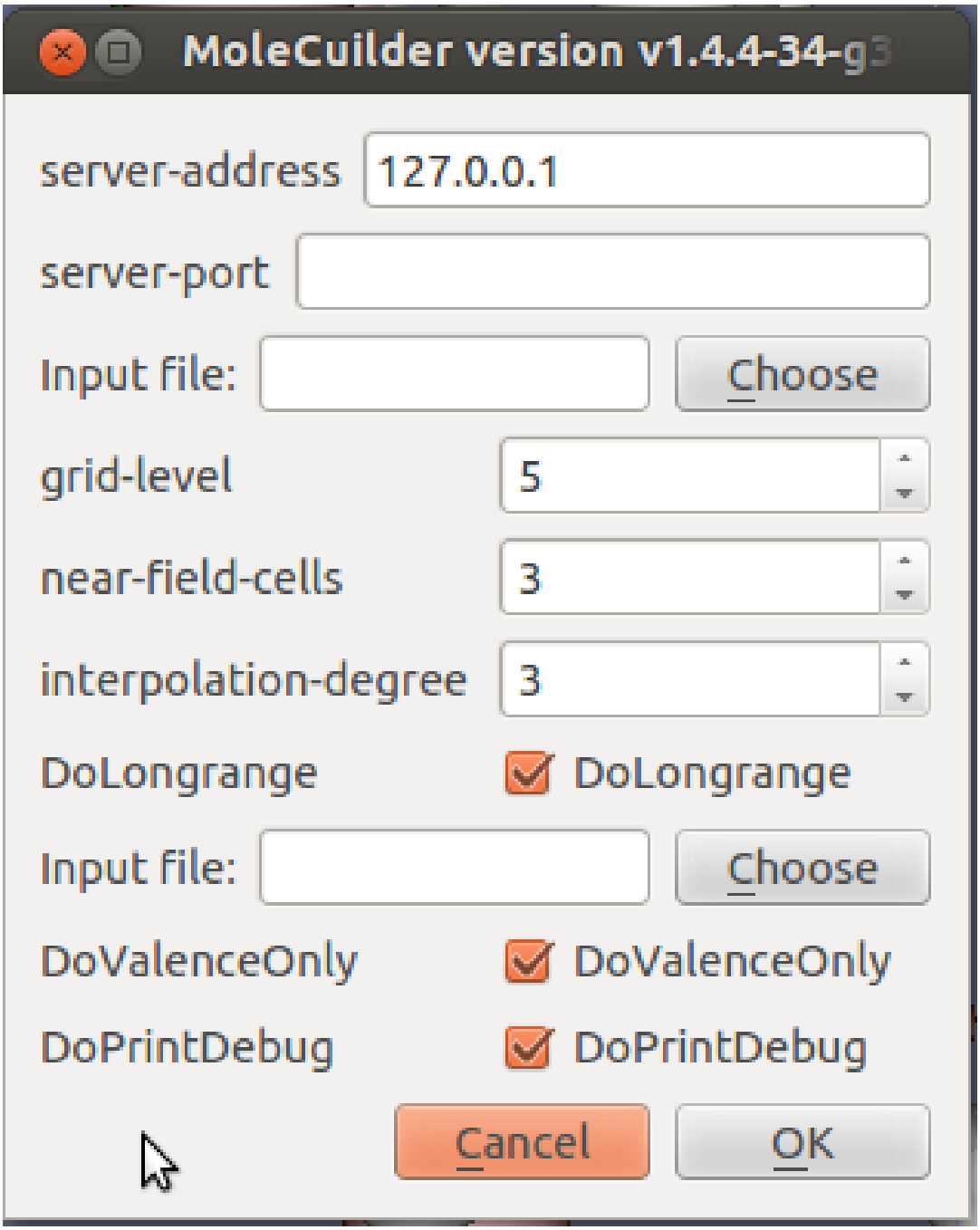


Elements are picked from a pull-down box where all known elements are listed.

In this dialog you also notice that a tooltip is given, briefly explaining what the action does.

Complex query

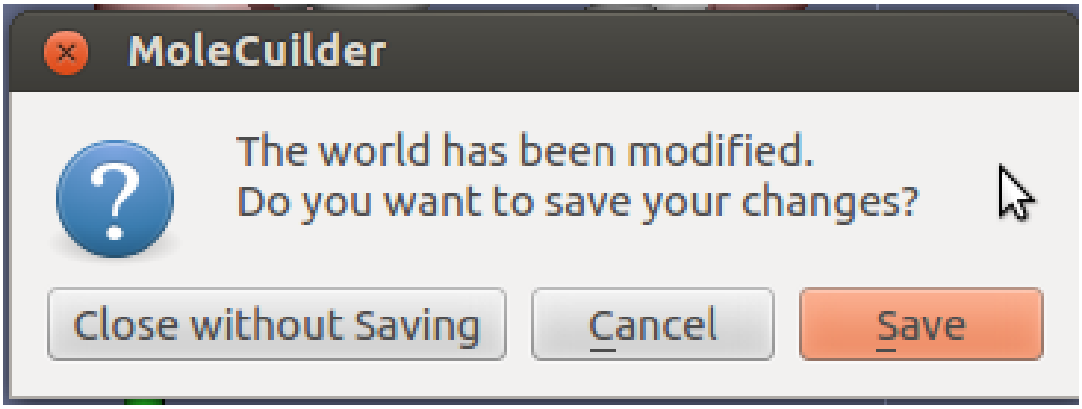
Figure 3.4. Screenshot of a complex dialog consisting of multiple queries



Here we show a more complex dialog. It queries for strings, for integer values (see the increase/decrease arrows), for booleans and for files (the "choose" buttons opens a file dialog).

Exit query

Figure 3.5. Screenshort showing the exit dialog



Finally, we show the dialog that will pop up when exiting the graphical interface. It will ask whether it should store the current state of the system in the input file or not. You may cancel the exit, close without saving or save the current state.

Python interface

Last but not least we elaborate on the python interface. We have already discusses this interface to some extent. The current session, i.e. the queue of actions you have executed, can be stored as a python script and subsequently executed independently of the user interface it was created with. More generally, MoleCuilder's Actions can be executed within arbitrary python scripts where prior to its execution a specific module has to be loaded, enabling access to MoleCuilder's actions from inside the script.

MoleCuilder's python module is called *pyMoleCuilder*. It is essentially a library that can be imported into python just as any other module. Let us assume you have started the python interpreter and you have added the containing folder of the `pyMoleCuilder` library to the `PYTHONPATH` variable.

```
import pyMoleCuilder as mol
```

Subsequently, you can access the help via

```
help(mol)
```

This will list all of MoleCuilder's actions with their function signatures within python as contained in the module *pyMoleCuilder* named as **mol** in the scope of the currently running interpreter. Note that the function names are not the names you know from the command-line interface, they might be called `WorldChangeBox(...)` or alike. However, they are related by a certain naming system. The first word is identical to the menu name it resides in the text or graphical interface. Then its followed by the actual name of the action that is similar to the command-line token.

Let's try it out.

```
print mol.CommandVersion()
```

This will state the current version of the library.

Go ahead and try out other commands. Refer to the documentation under the command-line interface and look up the function name via help.

You can freely mix calls to the pymolecuilder module and other python commands.

Note

However, be aware that all Actions are executed in another thread, i.e. run in parallel. That means that a pymolecuilder command is not necessarily finished when python steps on to the next line!

In order to make python wait for the Actions to finish before stepping, there is the special wait() command.

```
mol.MoleculeLoad("...")
status = mol.wait()
```

This will continue first only after the molecule has been fully loaded. Moreover, wait returns whether the actions contained executed succesfully.

Warning

These wait()s will have no effect if the python script is loaded via the "load-session" command inside a User Interface (command-line, GUI, ...) as this would cause the queue to wait indefinitely, namely till the load-session itself would have finished.

Therefore, more complex python scripts need to be called with python and a set PYTHONPATH as described above.

Note

It is *strongly* recommended to always use named arguments in python scripts when employing MoleCuilder commands. As commands tend to have many arguments, it is very easy to mix them up as python is not a strongly typed language.

Chapter 4. Conclusions

This ends this user guide.

We have given you a brief introduction to the aim of the program and how each of the four interfaces are to be used. The rest is up to you.

Tutorials and more information is available online, see MoleCuilder's website [http://www.molecuilder.com/].

Be aware that in general knowing how the code works allows you to understand what's going wrong if something's going wrong.

Thanks

Huge thanks go out to Saskia Metzler who was patient enough to let me sit next to her while riding ten hours in a bus to Berlin as I was writing the very first version of this guide.