ProtoMS Documentation

Release 3.4

C. J. Woods, J. Michel, M. Bodnarchuk, S. Genheden, R. Bradshaw, G. Ross, C. Cave-Ayland, A. I. Cabedo Martinez, H. Bruce-Macdonald, J. Graham, M. Samways

Aug 02, 2018

CONTENTS

CHAPTER

ONE

INTRODUCTION

ProtoMS is short for "Prototype Molecular Simulation", and is a software package that was originally designed by Dr. Christopher Woods to perform protein-ligand binding free energy calculations during his PhD. Dr. Julien Michel and Dr. Michael Bodnarchuk latter added numerous features and used the program extensively during their PhDs. Dr. Samuel Genheden, Dr. Richard Bradshaw, Dr. Gregory Ross, Dr. Chris Cave-Ayland, Dr. Ana Cabedo Martinez, Hannah Bruce-Macdonald, James Graham and Marley Samways have since then made numerous additions to the code among them a complete revision of the tools used to setup and analyse the simulation results.

The program is routinely used by several members of the research group of Professor Jonathan Essex for the development of new techniques to perform free energy calculations. This document has been written to try and explain how to use ProtoMS.

The user manual has been written as a reference manual, with extensive hyperlinking to allow you to quickly dip in and out to find the information you need. While you could read it from start to end, it would be a boring and repetitive read and you probably wouldn't learn much! We recommend that you engage with the tutorials that come with ProtoMS. You can then use the links in those descriptions to dip in and out of the user manual, thus obtaining a more detailed knowledge of how the examples, and thus ProtoMS, work.

1.1 Formatting

The following formats are used throughout this document. Program commands or contents of files will be written in monotype, e.g.

```
temperature float
```

where float is a floating point option to the command. If this option is given a value (e.g. 25.0), then it is written like this

```
temperature 25.0
```

The following options are standard to many commands

- **float** A floating point number.
- **integer** An integer. Most integer options given to ProtoMS are positive integers, greater than 0. This will always be made clear with the command.
- **logical** A logical, true or false option. Possible values for this option are true or false, yes or no or on or off, depending on your personal preference.
- **filename** This is the name of a file. Note that while ProtoMS is mostly case insensitive, file handling is dependent on the operating system you are using, so the filenames may be case sensitive. UNIX/Linux are examples of operating systems where case is important, while case is not important for Windows.

CHAPTER

TWO

CHANGE LOG

This section describess the new and updated functionality present in version 3.4.0 of ProtoMS.

2.1 GCAP Simulations

ProtoMS is now able to perform grand canonical alchemical perturbation (GCAP) simulations, where ligand perturbations can be performed with GCMC water sampling. These can be set up using `protoms.py -s gcap_single` or `protoms.py -s gcap_dual`, for single and dual topology calculations, respectively. These can either be performed at a range of B values, known as surface-GCAP (two-dimensional simulation in B and λ) or at the equilibrium B value, which is single-GCAP. These new simulation methods come with a new analysis tool `calc_gcap_surface.py` which is able to calculate the free energy from the two-dimensional GCAP surface.

2.2 Grand Canonical Integration

More automation has been introduced to `calc_gci.py` to calculate the appropriate number of steps for the titration fit. The water occupancy at the equilibrium B value is also returned.

2.3 Free Energy Calculations

Updates have been made to the analysis of free energy calculations. `calc_ti_decomposed.py` allows the different contributions to the free energy changes to be extracted. This is useful to understand where changes in relative free energies are arising from. `calc_dg_cycle.py` allows the cycle closure and free energy changes of various legs to be determined for a set of free energy perturbations.

Additionally, a 'partial' implementation of dual topology calculations is now available. If the change between two ligands in a relative free energy calculation is too large for single topology, but dual topology would not be appropriate, it is now possible to apply softcore potentials to only the changing groups, whilst leaving the common structure unchanged, reducing noise in the calculated free energy.

2.4 protomslib

The underlying class structures and many of the fundamental functions of the Python layer of ProtoMS have now been restructured into an importable module, `protomslib`. This also allows for support of Python $\geq$ 3.5 (retaining support for Python 2.7) with the setup and analysis scripts.

2.5 Restarting Simulations

The writing of restart files in ProtoMS has now been significantly improved, such that manual editing of the `.cmd` file is no longer required to restart a simulation which was terminated early or did not complete. Now, the simulation can be restarted where it left off by simply re-running the original command (something like `$PROTOMSHOME/protoms3 run.cmd`). ProtoMS will read the restart file written out and will know where to continue the simulation from.

2.6 Water Clustering

The clustering of GCMC waters with `calc_clusters.py` has been changed to prevent two waters observed in the same frame from being placed in the same cluster. This previously allowed for cluster occupancies of greater than 100%, and has now been corrected. The clustering should also now be less sensitive to the distance cutoff used.

CHAPTER

THREE

COMPILATION AND INSTALLATION

The ProtoMS package supplies the following files and directories;

- **CMakeLists.txt** This file configures ProtoMS prior to compiling.
- **data** This directory contains a number of useful files, e.g. pre-equilibrated boxes and some template files
- **doc** This directory contains documentation
- **README** File that contains brief installation instructions for ProtoMS, and any last minute addendums or errata that arrived too late to make it into the manual!
- **parameter** This directory contains all of the standard parameter files that describe the standard forcefields implemented in ProtoMS.
- **protoms.py** A tool to setup common ProtoMS calculations
- **python** This directory contains the python library component of ProtoMS
- **src** This directory contains all of the source code for the main program
- **tools** This directory contains numerous useful scripts to setup and analyse ProtoMS simualtions.
- **tutorial** This directory contains a number of examples that demonstrate applications of ProtoMS.

3.1 Programming Language

ProtoMS is written in slightly extended Fortran 77. The extensions used are

- The maximum line length is up to 132 characters, rather than 72.
- Variable, subroutine and function names are greater than 6 characters.
- `do/enddo` loops are used rather than `do/continue`.
- Fortran `include` is used to include the contents of other files.
- The `flush`, `getarg` and `getenv` non-standard intrinsic functions are used.
- ProtoMS performs string manipulation using the `len` function. In addition, the string manipulation assumes the same string handling behaviour as the GNU Fortran compiler (g77), so there is the possibility of strange formatting bugs when using different compilers.
- The `Date` and `Time` Fortran 90 intrinsic subroutine is used to get the current time. This is used to provide a default seed to the random number generator. This can be removed by commenting out the relavant lines in `getoptions.F`, though you will need to provide a random number seed manually.

3.2 Requirements

The *MC program* has the following requirements:

- Fortan compiler, GNU (https://gcc.gnu.org/) or Intel recommended
- Python (https://www.python.org/), required to compile and run ProtoMS
- CMake (http://www.cmake.org/), required to compile ProtoMS

Optional:

- MPI, recommended OpenMPI (http://www.open-mpi.org/) or MPICH (https://www.mpich.org/) - some functions will be unavailable without MPI

The *ProtoMS tools* have the following requirements:

- Python, version 2.7, version 3.5 or newer
- NumPy (http://www.numpy.org/)
- SciPy (http://www.scipy.org/)
- Matplotlib (http://www.matplotlib.org/)
- pymbar (https://github.com/choderalab/pymbar)

Optional:

- AmberTools (http://www.ambermd.org/) : Required to parameterise small molecules

The tools are written in python (https://www.python.org/) and are compatible with version 2.7 as well as version 3.5, 3.6 and 3.7.

A docker image that contains all of the installation dependencies is available. See *Using Docker* below.

3.3 Installing ProtoMS

ProtoMS has been written using the GNU Fortran compilers (https://gcc.gnu.org/), on the Linux operating system. ProtoMS is thus known to work well with this compiler and Linux. ProtoMS has also been compiled and tested using the Intel Fortran Compiler. ProtoMS has been compiled with other compilers but not extensively tested. It is therefore strongly advised to use either GNU or Intel compilers with ProtoMS.

You also need an MPI package to perform simulations that require multiple processes, e.g. replica exchange. Such libraries should be available on most modern computers and clusters. The MPI compilers in the GNU package is called mpiff77. ProtoMS has been compiled with both OpenMPI (http://www.open-mpi.org/) and MPICH (https://www.mpich.org/). However, note that this is *not* a requirement any more. ProtoMS will compile without OpenMPI, but you won't be able to run for instance replica exchange.

Building ProtoMS is done with *cmake* http://www.cmake.org/, thus you need this package installed on your machine. Before proceeding it is important that your environment is properly configured. In particular, since version 3.4, ProtoMS installs the Python package *protomslib* into your python environment. If you are using a virtual environment this must be activated so that cmake can locate the correct python interpreter. To prepare the build, type the following in a terminal from the root directory of the code:

```
mkdir build

cd build

cmake ..
```

At this point you should check the output of cmake. Unless you're expecting it not to, cmake should have found an appropriate Fortran compiler, MPI library and Python interpreter. Check that the paths and versions of these correspond to those you expect. If they do not, see *Customising the Build* for details on how to customise these. Also note that if cmake has found the system python interpreter (usually /usr/bin/python) it will attempt to install protomslib into a system location requiring root access. Again *Customising the Build* covers how to change the Python installation target. If you're happy with what cmake has found then type:

```
make install
```

and *cmake* will perform the necessary checks before it continues with the installation of ProtoMS. The executable will be placed in the top level of the folder hierarchy.

In order for ProtoMS to find the relevant parameter files it is necessary to set the environmental variable `$PROTOMSHOME` to the installation directory of ProtoMS. This variable is used as a shortcut in the tutorials and by the Python tools. ProtoMS is also able to substitute this variable when it is used in ProtoMS command files.

Once building is complete it is highly recommended to run the test suite that comes with ProtoMS to test that the build was successful. From the build directory created above simply type:

```
ctest -V
```

All tests should be expected to pass and the above command will provide detailed output. The most common reason for failures is the need to set the correct environment variables. Notably `$AMBERHOME` for the setup tests and `$PROTOMSHOME`, as described above. Another reason for occasional failures is slight formatting and rounding differences between compilers, this can lead to values differing at the final decimal place in results files and such failures can be safely ignored.

3.4 Customising the Build

Tips on using cmake

The job of cmake is to attempt to locate all of the necessary dependencies for the installation and create a Makefile that will compile ProtoMS. It searches your system for the required components and sets a number of internal variables that store their locations. After being run cmake stores its output in the build directory in a file called CMakeCache.txt. This can be useful after the fact to check which dependencies were found but equally if being run subsequently cmake will prefer to use cached values instead of updating dependencies. For this reason it can be a good idea to delete CMakeCache.txt if you find you need to run cmake more than once or cmake does not appear to be behaving as expected.

Manually specify cmake variables

The locations that cmake will search for dependencies are quite comprehensive, however they are also dependent on the system in use and the value of current environment variables. Thus cmake may not be able to find the required libraries even if they're present in your system or may find the wrong versions. To coerce cmake into finding the relevant dependencies you can try:

1. Setting environment variables - The $PATH environment variable is checked by cmake for relevant executables e.g. gfortran, mpirun. Prepending to or rearranging entries in the PATH makes dependencies discoverable by cmake. The FC environment variable is a standard method for manually specifying the Fortran compiler.
2. Manually setting cmake variables - Whilst cmake attempts to automatically discover correct values for dependencies you may find that setting them manually is easier. This can be performed interactively using the `ccmake` utility. If you execute `ccmake ..` from the build directory you will be presented with a interface showing the current value of cmake variables. Press `t` to see more values.

You can edit values from this menu before pressing `c` to configure (any problems should be flagged by cmake here) and `g` to generate a new Makefile and exit.

3. Manually setting cmake variables on the command line - If you prefer the value of any cmake variable can be specified directly from the command line. The `-D` flag to cmake can be used repeatedly for this purpose. For instance - `cmake -DCMAKE_Fortran_COMPILER=gfortran ..` - sets the value of the variable CMAKE_Fortran_COMPILER to gfortran. You can use `ccmake` to determine the names of variables to set.

Installation of protomslib

You can customise the installation of the python library component by specify a value for the cmake variable PYTHON_INSTALL_OPTIONS (see above). The value of this variable will be appended like so to the command below which is executed by cmake:

```
python setup.py install $PYTHON_INSTALL_OPTIONS
```

To see the available options you can run:

```
python $PROTOMSHOME/python/setup.py install --help
```

The most frequently useful options are `--user`, that requests an installation into `$HOME/.local`, and –prefix that allows an installation root directory to be specified manually.

3.5 Using Docker

Version 3.4 of ProtoMS is also available via docker. Downloading and running the image can be accomplished easily with the command:

```
docker run -it jessexgroup/protoms:3.4
```

The image is based on the python:3.6.6 image with additional installation of the relevant python dependencies as well as amber tools 18. To construct your own docker images from scratch see `Dockerfile_test` and `Dockerfile` in the root ProtoMS directory and the instructions therein. This will allow you to use newer versions of the dependencies than are available via the public image.

CHAPTER

FOUR

DESIGN OF PROTOMS

ProtoMS is a powerful simulation program that is capable of being used in many different ways. ProtoMS was originally designed to perform Monte Carlo free energy calculations on protein-ligand systems, so a lot of the terminology and ideas associated with ProtoMS derive from protein-ligand Monte Carlo methodology. While the code was originally designed with this use in mind, the framework is sufficiently flexible to allow the study of a wide range of different systems, using a wide range of simulation methodology.

At the core of ProtoMS are four central concepts;

- **Proteins/Solutes/Solvents/GCSolutes** ProtoMS divides all molecules to be simulated into 'proteins', 'solutes' and 'solvent'.
- **Classical Forcefields** ProtoMS Uses a generic classical forcefield to calculate the energy of the molecules. This forcefield may be specialised such that ProtoMS is able to implement a wide range of modern molecular mechanics forcefields.
- **Perturbations** ProtoMS provides support for free energy calculations by allowing forcefields and geometries to be perturbed using a λ coordinate. The forcefield for any protein, solute or solvent may be perturbed, and the geometry of any solute may be perturbed.
- **Generic Moves** ProtoMS is designed around the concept a 'move'. The move can do anything, from a Monte Carlo translation of solvent to a docking type move on a solute. A simulation is constructed by stringing a collection of moves together.

4.1 Proteins / Solutes / Solvents / GCSolutes

ProtoMS divides all of the molecules loaded within a system into solvents, GCsolutes, solutes and proteins

- **solvents** A solvent is any rigid molecule. Solvents may only be translated and rotated, and by default, 75000 solvent molecules may be loaded, each consisting of up to 10 atoms. Solvent molecules do not have to be small - a rigid lipid molecule could be modelled as a solvent. There is no requirement for the solvents loaded in a system to be the same. Indeed every solvent loaded could be a different type of molecule!
- **GCsolutes** Like a solvent molecule, GCsolutes are rigid. They have the same properties as previously described for solvents, except GCsolutes are restrained to a defined region in the simulation.
- **solutes** A solute is any flexible molecule. Solutes can be translated and rotated, and change their internal geometry. By default 60 solutes, each composed of 10 residues, each composed of 100 atoms may be loaded simultaneously. Solute molecules are described using z-matrices, thus a solute molecule is perhaps what you would be most familiar with from other Monte Carlo simulation programs. Note that you can describe a protein molecule as a solute, and that you do not need to load it up as a 'protein'.
- **proteins** A protein is any flexible chain molecule (polymer). A protein is composed of a linear chain of residues, with interresidue bonds connecting one residue to the next. By default, ProtoMS can load up to 3 proteins simultaneously, each protein consisting of 1000 residues, each consisting of up to 34 atoms.

Solvents

Solvents are loaded into ProtoMS from PDB files (see section *Solvent File*). Each solvent molecule is identified by its residue name (the fourth column in the PDB file), e.g. ProtoMS identifies the TIP4P solvent with the residue name 'T4P'. ProtoMS loads the coordinates of the solvent from the PDB file, and then assigns the parameters for the solvent from a solvent template (see section *Templates*). The solvent template contains the information necessary to identify all of the atoms in the solvent molecule and to assign forcefield parameters to each atom. Note that this version of ProtoMS uses the coordinates of the solvent molecule that are present in the PDB file. ProtoMS does not yet have the capability to modify these coordinates to ensure that the internal geometry of the solvent is correct for the solvent model. This means that as solvents are only translated and rotated, the internal geometry of the solvent molecule loaded at the start of the simulation will be identical to that at the end of the simulation.

GCSolutes

Like solvents, GCsolutes are loaded into ProtoMS from PDB files (see section *GCsolute File*). Each GCsolute molecule is identified by its residue name (the fourth column in the PDB file). ProtoMS loads the coordinates of the GCsolute from the PDB file, and then assigns the parameters for the GCsolute from a GCsolute template (see section *Templates*). This template contains the information necessary to identify all of the atoms in the solvent molecule and to assign forcefield parameters to each atom. Alongside translational and rotational moves, the intermolecular energy between the GCsolute and the system can be sampled.

Solutes

Solutes are also loaded into ProtoMS from PDB files (see section *Solute File*). Each solute molecule is identified by its solute name, which is given in the HEADER record of the PDB file. ProtoMS obtains the coordinates of the solute from the PDB file, and will then find a solute template that matches this solute name (see *Templates*). The solute template is used to build the z-matrix for the solute, and to assign all of the forcefield parameters. The solute template is also used to assign the connectivity of the solute and to define the flexible internal coordinates. The solute molecule is constructed using the z-matrix, with the reference being three automatically added dummy atoms, called 'DM1', 'DM2' and 'DM3', all part of residue 'DUM'. These dummy atoms are automatically added by ProtoMS at the geometric center of the solute, as a right angled set of atoms pointing along the major and minor axes of the solute.

Proteins Proteins are loaded into ProtoMS via PDB files (see section *Protein File*). Each PDB file may only contain a single protein chain. ProtoMS constructs the linear chain of molecules based on the order of residues that it reads from the PDB file, and will ignore the residue number read from the PDB file. This means that you must ensure that you have the residues ordered correctly within the PDB file. ProtoMS assigns to each residue both a chain template (see section *Templates*), that describes the backbone of the residue, and a residue template (see section *Templates*), that describes the sidechain. The residue template is located based on the name of the residue given in the fourth column in the PDB file (e.g. 'ASP' or 'HIS'). The chain template is located based on the chain template associated with the residue template for the position of the residue within the chain. For example, residue 'ASP' has a standard amino acid backbone chain template if this residue was in the middle of the chain, an NH+ capped backbone chain template 3 if this was the first residue of the chain (and thus at the n-terminus), and a CO– capped backbone chain template 2 if this were the last residue of the chain (and thus at the c-terminus). If the protein consisted of only one residue, then the zwitterionic amino acid chain template would be used for 'ASP'.

ProtoMS obtains the coordinates of each residue from the PDB file, and will then use the residue and chain templates to build the z-matrix for each residue, and to assign all of the forcefield parameters.

Proteins are moved in a different manner in ProtoMS compared to other Monte Carlo packages that are available. Each residue is moved independently, using both the internal geometry moves defined by the template z-matrix, and by backbone translation and rotation moves of the chain atoms (see figure above).

Four special backbone atoms (bbatoms) are identified in the chain-backbone of each residue. These atoms form the reference from which the rest of the residue atoms are built. These four atoms can be translated and rotated as a rigid unit via protein backbone moves (see figure above). As the rest of the residue is constructed from these bbatoms, the rest of the residue is thus also translated and rotated. Because the bbatoms are translated and rotated as a rigid unit, the internal geometry of these backbone atoms are held constant throughout the simulation. This means that the internal geometry of the bbatoms is taken from the PDB file, and may not be modified by the chain or residue templates. It is

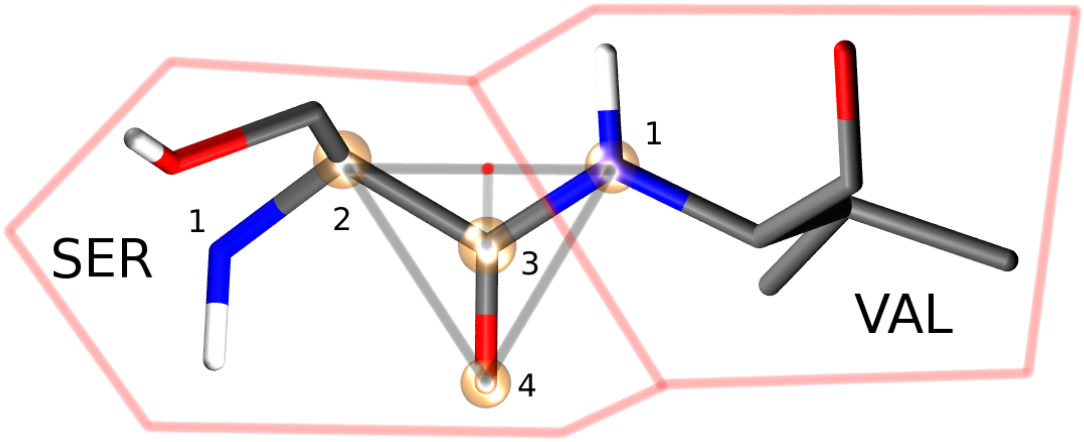

Fig. 1: Four atoms from each protein residue are designated as backbone atoms (bbatoms). For most residues these atoms are the N, CA, C and O atoms respectively. The four backbone atoms for two neighbouring residues are shown above. The protein backbone move moves the last three bbatoms of one residue and the first bbatom of the next residue. This is because the moves assumes that these four bbatoms form a rigid triangle (as is shown by the grey lines). The four atoms are translated and rotated as a rigid triangle, with the origin of rotation of the triangle centered on the intersection of the vector between bbatoms 2 and 1, and the vector between bbatoms 3 and 4 (marked as a red dot directly above the C=O bond). Because this triangle is translated and rotated as a rigid unit, all atoms connected to the atoms of this triangle will also be translated and rotated as a rigid unit.

also not possible to build missing bbatoms, so they must all be present in the PDB file.

Once the coordinates and z-matrices of each residue have been assigned, interresidue bonds are added between the first bbatom of each residue and the third bbatom of the previous residue (e.g. for 'ASP', bonds would be added from the 'N' atom of the 'ASP' residue to the 'C' atom of the preceeding amino acid residue). If the length of this bond is less than 4 A then this bond is added as a real bond, and its energy is evaluated as part of the forcefield. However, if the length is greater than 4 A, then this bond will be added as a dummy bond, and a warning message output. This is useful in cases where you wish to load up a protein scoop, e.g. from around the active site. This option should be used with care in conjunction with backbone moves.

Table 1: Table 1.0 The default value of the maximum number of proteins, GCsolutes, solutes and solvents that may be loaded simultaneously by ProtoMS. These values may be changed by editing the `dimensions.inc` file located in the src directory, and recompiling ProtoMS.

Parameter	Description	Values
MAXPROTEINS	Maximum number of proteins	3
MAXRESIDUES	Maximum number of residues per protein	1000
MAXSCATOMS	Maximum number of atoms per protein residue	30
MAXSOLUTES	Maximum number of solutes	60
MAXSOLUTERESIDUES	Maximum number of residues per solute	10
MAXSOLUTEATOMSPERRESIDUE	Maximum number of solute atoms per residue	100
MAXSOLVENTS	Maximum number of solvent molecules	75000
MAXSOLVENTS	Maximum number of GCsolute molecules	75000
MAXSOLVENTATOMS	Maximum number of atoms per solvent	10

Limits

ProtoMS is written using slightly extended Fortran 77 (see *Programming Language*). This means that the maximum

numbers of loaded proteins, solutes and solvents has to be set at compile time. Table 1.0 gives the default values for the maximum number of proteins, solutes and solvents. Please note that you may change these numbers to fit the system that you are interested in, e.g. if you were investigating a single protein in a lipid bilayer then you may choose to model the lipid as a solute (thus requiring a large increase in the number of solute molecules, but a decrease in the number of solute residues), and you could reduce the maximum number of protein molecules to one. By balancing the numbers of protein, solutes and solvents you should find that you are able to load up the system that you want to simulate.

4.2 Classical forcefields

ProtoMS was designed to perform simulations using a range of different molecular mechanics (MM) forcefields. To achieve this aim, a generic forcefield has been implemented, and this can be specialised into a specific, traditional forcefield. Specifically, ProtoMS supports the use of the Amber ff99, ff99SB and ff14SB protein forcefields, along with OPLS 96. The General Amber Forcefield (GAFF) is used for solutes, whilst various solvent forcefield models including TIP3P, TIP4P, SPC and SPC/E can be used.

The forcefield in ProtoMS is comprised of several terms;

Intermolecular Potential

An intermolecular potential acts between all molecules within the system. The intermolecular potential between a pair of molecules, *A* and *B*, $U_{molecule}(A, B)$, with *A* consisting of n_A atoms and *B* consisting of n_B atoms, is formed as the sum of the non-bonded potential, $U_{nb}(i, j)$ between each pair of atom sites, *i* and *j*, between the two molecules, scaled by a constant, *scl*, e.g.

$$U_{molecule}(A, B) = scl(R) \times \left(\sum_{i=1}^{n_A} \sum_{j=1}^{n_B} U_{nb}(i, j) \right) \tag{4.1}$$

where *R* is the shortest distance between a pair of atom sites between the molecules. The scaling factor is set according to

$$R \geq r_{cut} \rightarrow scl = 0.0$$

$$r_{cut} - r_{feather} \leq R \leq r_{cut} \rightarrow scl = \frac{r_{cut}^2 - R^2}{r_{cut}^2 - (r_{cut} - r_{feather})^2}$$

$$R \leq r_{feather} \rightarrow scl = 1.0,$$

where r_{cut} and $r_{feather}$ are the non-bonded cutoff and feather parameters.

The non-bonded potential between the pair of atoms is evaluated as the sum of the Coulombic and Lennard-Jones (LJ) potentials between the atoms,

$$U_{nb}(i, j) = \frac{q_i q_j}{4\pi\epsilon_0 r(i, j)} + 4\epsilon_{ij} \left[\left(\frac{\sigma_{ij}}{r(i, j)} \right)^{12} - \left(\frac{\sigma_{ij}}{r(i, j)} \right)^{6} \right], \tag{4.2}$$

where q_i and q_j are the partial charges on the two atom sites, *r(i, j)* is the distance between the atom sites, ϵ_0 is the permittivity of free space and σ_{ij} and ϵ_{ij} are the Lennard Jones parameters for the atom site pair *i* and *j*. The LJ parameters for an atom site pair are calculated as the average of the LJ parameters for the same site pair.

Either the arithmetic average is used, or the geometric average is used, e.g.

$$\sigma_{ij} = 0.5 \times (\sigma_{ii} + \sigma_{jj}). \tag{4.3}$$

$$\epsilon_{ij} = \sqrt{\epsilon_{ii} \times \epsilon_{jj}}. \tag{4.4}$$

The AMBER family of forcefields use the arithmetic average for σ, and the geometric average for ϵ, while the OPLS family of forcefields use the geometric average for both parameters. The intermolecular potential is formed as the sum of the non-bonded potential over all pairs of atom sites. It should be noted that an atom site does not necessarily need to lie at the center of each atom, and it may lie between atoms, or at the location of any lone pairs. Individual atoms may possess many atom sites, or even no atom sites.

Bond Potential

A bond potential acts over all of the explicitly added, non-dummy bonds within a molecule. ProtoMS makes no attempt to find any implicit bonds within a molecule, and it is not possible to add a bond between atoms of different molecules. The energy of each bond, U_{bond} , is evaluated according to

$$U_{bond}(r) = k_{bond}\big(r - r_0\big)^2, \tag{4.5}$$

where *r* is the bond length, k_{bond} is the force constant for the bond, and r_0 is the equilibrium bond length. The total bond energy of a molecule is the sum of the bond energies for all of the bonds within the molecule, and the total bond energy of the system is the sum of the bond energies for each of the molecules in the system.

Angle Potential

An angle potential acts over all angles between atoms that are connected by non-dummy bonds, and over all non-dummy angles that have been explicitly added to the molecule. The energy of each angle, Uangle , is evaluated according to

$$U_{angle}(\theta) = k_{angle}\big(\theta - \theta_0\big)^2, \tag{4.6}$$

where θ is the size of the angle, k_{angle} is the force constant for the angle, and θ_0 is the equilibrium angle size. The total angle energy of a molecule is the sum of the angle energies for each of the angles within the molecule, and the total energy of the system is the sum of the angle energies for each of the molecules in the system.

Urey-Bradley Potential

A Urey-Bradley potential may act between the first and third atoms of some of the angles that are evaluated for the angle potential. If this is the case, then a Urey-Bradley energy is added onto the angle energy. The Urey-Bradley energy, U_{uby}, is evaluated according to

$$U_{uby}(x) = k_{uby}\big(x - x_0\big)^2, \tag{4.7}$$

where *x* is the distance between the first and third atoms, k_{uby} is the Urey-Bradley force constant, and x_0 is the equilibrium distance.

Dihedral Potential

A dihedral potential acts over all dihedrals between atoms that are connected by non-dummy bonds, and over all non-dummy dihedrals that have been explicitly added to the molecule. Such explicitly added dihedrals may be used to add improper dihedrals that maintain the stereochemistry of chiral centers. The energy for each dihedral, $U_{dihedral}$, is formed as the sum of *n* cosine terms,

$$U_{dihedral}(\phi) = \sum_{i=1}^{n} k_{i1}\big[1.0 + k_{i2}\big(cos(k_{i3}\phi + k_{i4})\big)\big], \tag{4.8}$$

where k_{i1} to k_{i4} are dihedral parameters and ϕ is the size of the dihedral. The total dihedral energy of a molecule is the sum of the dihedral energies for each of the dihedrals in the molecule, and the total dihedral energy of the system is the sum of the dihedral energies of each of the molecules.

Intramolecular non-bonded Potential

An intramolecular non-bonded potential acts between all intramolecular pairs of atoms that are either not connected by a non-dummy bond, or are not both connected to a third atom by a non-dummy bond. To make this more clear, if two atoms are connected by a non-dummy bond then they are said to be 1-2 bonded. If two atoms are both connected

to a third atom by non-dummy bonds, then they are said to 1-?-3, or 1-3 bonded. Similarly, if the pair of atoms are connected together via two atoms via non-dummy bonds, then they are said to be 1-?-?-4, or 1-4 bonded. An intramolecular non-bonded potential does not act over 1-2 or 1-3 bonded pairs within a molecule, but does act over 1-4 bonded pairs and above. Note that ProtoMS only looks at the non-dummy bonds between atoms, and will not consider whether or not there are non-dummy angles, Urey-Bradley or dihedral terms involving these atoms.

The intramolecular non-bonded potential of a molecule, U_{intra} is the sum of the non-bonded energy between all 1-5 and above pairs of atoms within the molecule, plus the sum of the non-bonded energy between all 1-4 atoms scaled by a 1-4 scaling factor, e.g.

$$U_{intra} = \sum_{\text{1-5+ i j pairs}} U_{coul}(i,j) + U_{lj}(i,j) + \sum_{\text{1-4 i j pairs}} scl_{coul} U_{coul}(i,j) + scl_{lj} U_{lj}(i,j), \tag{4.9}$$

where

$$U_{coul}(i,j) = \frac{q_i q_j}{4\pi\epsilon_0 r}, \tag{4.10}$$

and

$$U_{lj}(i,j) = 4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}}{r}\right)^{12} - \left(\frac{\sigma_{ij}}{r}\right)^{6}\right]. \tag{4.11}$$

Equations (4.10) and (4.11) are the Coulomb and Lennard Jones equations, as seen in the intermolecular potential in equations (4.1) and (4.2). scl_{coul} and scl_{lj} are the Coulomb and Lennard Jones scaling factors.

Generalized Born Surface Area potential

While free energy simulations are usually conducted in explicit solvent, ProtoMS supports Generalized Born Surface Area (GBSA) implicit solvent models. Relatively few free energy implicit solvent studies have been conducted and such option should be tested carefully before embarking onto expensive free energy simulations. The GBSA theory assumes that the total solvation free energy of a molecule A is a sum of a polar and non-polar energy term:

$$\Delta G_{solv} = \Delta G_{pol} + \Delta G_{nonpol} \tag{4.12}$$

The second term, is simply proportional to the solvent accessible surface area (SASA) of the molecule, times a parameter that depends on the atom types present in the molecule. The first term is more complex and derived from the following equation :

$$\Delta G_{pol} = -\frac{1}{2}\left(\frac{1}{\epsilon_{vac}} - \frac{1}{\epsilon_{solv}}\right)\sum_i \sum_j \frac{q_i q_j}{\sqrt{r_{ij}^2 + B_i B_j e^{\frac{-r_{ij}^2}{4B_i B_j}}}} \tag{4.13}$$

ϵ_{vac} and ϵ_{solv} are the dielectric constants of the vacuum and the solvent respectively, q_i the atomic partial charge of atom *i*, r_{ij} the distance between a pair of atoms *ij*, and B_i is the effective Born radius of atom *i*.

The effective Born Radius B_i is in essence the spherically averaged distance of the solute atom to the solvent. An accurate estimate of this quantity is essential to calculate high quality solvation free energies. It is however fairly complex to compute as it formalyl involves an integral over the position of all the atoms in the system. While numerical techniques can calculate such value, they are too slow to be of practical use in a simulation. In ProtoMS, the effective

Born radii are calculated using the Pairwise Descreening Approximation (PDA) method.

$$\frac{1}{B_i} = \frac{1}{\alpha_i} - \frac{1}{2}\sum_{j \neq i}[\frac{1}{L_{ij}} - \frac{1}{U_{ij}} + \frac{r_{ij}}{4}(\frac{1}{U_{ij}^2} - \frac{1}{L_{ij}^2}) + \frac{1}{2r_{ij}} ln \frac{L_{ij}}{U_{ij}} + \frac{S_j^2 \alpha_j^2}{4r_{ij}}(\frac{1}{L_{ij}^2} - \frac{1}{U_{ij}^2})]$$

$$L_{ij} = 1 \quad if \quad r_{ij} + S_j\alpha_j \leq \alpha_i$$

$$L_{ij} = \alpha_i \quad if \quad r_{ij} - S_j\alpha_j \leq \alpha_i < r_{ij} + S_j\alpha_j$$

$$L_{ij} = r_{ij} - S_j\alpha_j \quad if \quad \alpha_i \leq r_{ij} - S_j\alpha_j$$

$$U_{ij} = 1 \quad if \quad r_{ij} + S_j\alpha_j \leq \alpha_j$$

$$U_{ij} = r_{ij} + S_j\alpha_j \quad if \quad \alpha i < r_{ij} + S_j\alpha_j$$

where r_{ij} is the distance between a pair of atoms *ij* and α_i is the intrinsic Born radius of atom *i*, that is, the Born radius that atom *i* would adopt if it was completely isolated. Finally S_j is a scaling factor which compensates for systematic errors introduced by this approximate Born radii calculation.

As the name says, the technique approximate the descreening (the extent to which a nearby atom j displaces a volume that would have otherwise been occupied by solvent) by a fast summation of pairwise terms. It is however not rigorous and has to be parameterised carefully to yield robust performance. The PDA method tend to systematically underestimate the Born radius of buried atoms because it incorrectly assign high dielectric constants to numerous small voids and crevices that exist between atoms in a protein and are not occupied by water. To increase accuracy, a re-scaling technique has been implemented.

$$\frac{1}{B_i} = \frac{1}{\alpha_i} - Itanh\big(\alpha\psi - \beta\psi^2 + \gamma\psi^3\big)$$

where *I* is the summation term from the PDA calculation, ψ, α, β and γ are parameters taken from the litterature.

The rescaling option has not been used extensively in ProtoMS and should be used with caution. It appears it may prove useful when simulation buried protein binding sites.

The GBSA force field implemented in ProtoMS was parameterised to be used with the AMBER99 and the GAFF force fields. While alternative force fields could be used, a loss of accuracy could be expected.

GBSA simulations are order of magnitude more efficient than explicit solvent simulations of small isolated molecules. However, they slow down rapidly when the size of the system increases. This is especially notable in Monte Carlo simulations where a small movement of part of a system formally warrants the computation the entire solvation energy of the system. This issue arises because the GBSA energy terms are not strictly pairwise decomposable. It is possible to use however different techniques to increase the speed of a GBSA simulation. Cutoffs in the calculation of the Born radii are introduced and in addition the update of pairwise GB energies can be skipped if the Born radii of either atoms have not changed more than a certain threshold value after a MC move. Because this option will introduce energy drifts, it is advised to periodically recalculate rigorously the GB energy. In addition, a more complex Monte Carlo move is implemented in ProtoMS. This option allows to conduct a simulation with a crude GBSA model and a low cutoff for the non bonded energy terms. Normally the predicted macroscopic properties would suffer from such crude treatment of intermolecular energies. However, periodically, a special acceptance test is employed to remove the bias introduced by the crude potential and ensure that the equilibrium density of states generated by the Monte Carlo simulation converges to the equilibrium density of states suitable for the standard biomolecular potential.

Actual speedups using either techniques are system dependent and optimisation of the different parameters can be a complex task. It is advised to use the default parameters described latter in the manual.

Caveats

ProtoMS implements this forcefield mostly as described. However there are a few shortcuts that are taken to improve the efficiency of the code. These shortcuts are based on the three-way split of the molecules of the system into solvents, solutes and proteins

- **solvents** As solvents are rigid, there is no need to evaluate any of the intramolecular potentials. ProtoMS thus only evaluates the intermolecular energy of solvent molecules.
- **solutes** ProtoMS evaluates the forcefield of solute molecules exactly as described, with no shortcuts.
- **proteins**. ProtoMS implements a protein as a chain of residues. As these molecules can be large, and typically larger than the non-bonded cutoff, ProtoMS implements the non-bonded cutoff differently for proteins. Instead of evaluating the non-bonded cutoff for the protein as a whole, ProtoMS implements a residue-based cutoff, with the cutoff scaling factors evaluated individually for each residue. Additionally, the intramolecular non-bonded energy is also scaled according to the non-bonded cutoffs given in equation (4.1). If you do not want to use residue based cutoffs, then it is possible to tell ProtoMS to use a molecule based cutoff, in which case the forcefield for proteins will be evaluated exactly as described with no shortcuts.

4.3 Perturbations

ProtoMS is capable of calculating the relative free energy of two systems. ProtoMS does this by perturbing one system into the other through the use of a λ-coordinate. If A and B are the two systems of interest, then the forcefield is constructed such that at λ = 0.0 the forcefield represents system A, at λ = 1.0 the forcefield represents system B, and at λ value inbetween, the forcefield represents a hybrid of A and B.

ProtoMS implements two methods of perturbing between systems A and B;

- **Single topology** System A is perturbed into system B by scaling the forcefield parameters such that the model morphs from A to B.
- **Dual topology** System A and B are simulated together, with λ scaling the total energies of A and B such that one system is turned off as the other is turned on.

Single Topology Calculations

ProtoMS assigns two sets of parameters to every single forcefield term; one parameter represents that term at $\lambda = 0.0$ (par_0), the other represents that term at $\lambda = 1.0$ (par_1). λ is used to linearly scale between these two parameters to obtain the value of the parameter at each value of λ (par_λ)

$$par_\lambda = (1.0 - \lambda) \times par_0 + \lambda \times par_1. \tag{4.14}$$

This equation is used to scale the charge, σ and ϵ parameters assigned to each atom site (see equations (4.1)), and the force constants (k_{bond}, k_{angle} and k_{uby}) and equilibrium sizes (r_0, θ_0 and x_0) for the bond, angle and Urey-Bradley terms (see equations (4.5), (4.6) and (4.7)). This equation is not used to scale the dihedral parameters, as the functional form of the dihedral potential is more complicated. Rather than scale the dihedral parameters, ProtoMS uses λ to scale the total energy of each dihedral;

$$U_{dihedral}(\phi)_\lambda = (1.0 - \lambda) \times U_{dihedral}(\phi)_0 + \lambda \times U_{dihedral}(\phi)_1, \tag{4.15}$$

where $U_{dihedral}(\phi)_0$ is the dihedral energy using the parameters for $\lambda = 0.0$, $U_{dihedral}(\phi)_1$ is the dihedral energy using the parameters for $\lambda = 1.0$, and $U_{dihedral}(\phi)_\lambda$ is the scaled dihedral energy at that value of λ.

Any and all parts of the forcefield can be scaled. This includes all of the forcefield parameters of any solutes, all of the parameters of any proteins, and all parameters of any solvent molecules. While this is very useful, and enables perturbations of any and all parts of the system, there are many cases where just changing the forcefield parameters is not sufficient to smoothly morph from one system into the other. There are many cases where the geometry of the molecules needs to be changed with λ. Fortunately ProtoMS provides this capability for solute molecules. Any internal coordinates that are part of the z-matrix of a solute molecule may perturbed with λ. Geometry variations are a

powerful tool as they allow for very complicated, yet very smooth transitions between two systems to be described. A good example of such a transition is the annihilation of the hydrogen atoms as a methyl group is morphed into a single hydrogen.

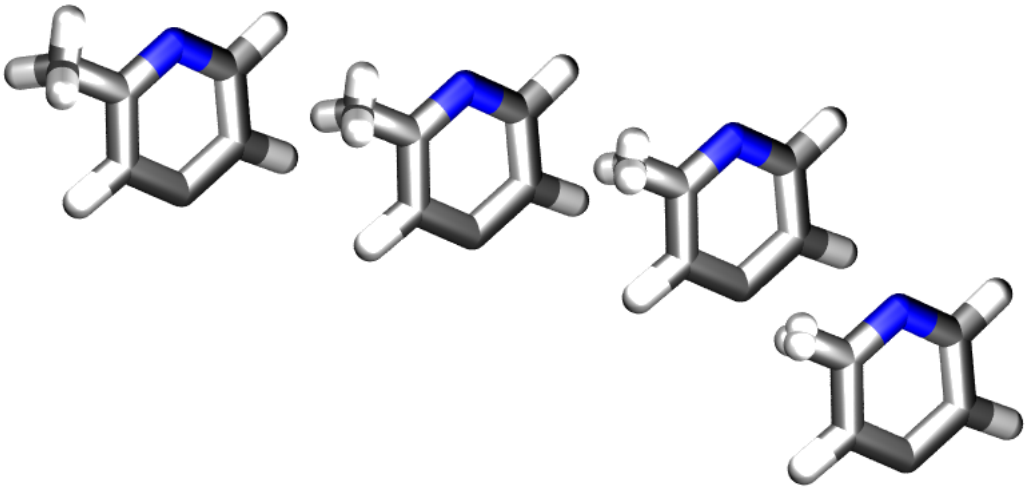

Fig. 2: Geometry variations allow for a smoother transition between two systems, for example here a methyl group is smoothly converted into a hydrogen.

As well as enabling smooth transitions between systems, geometry variations may be used to calculate potentials of mean force along structural coordinates.

Dual Topology Calculations

A dual topology method to calculate free energy changes is also available in ProtoMS. In the single topology method force field terms were linearly interpolated so that they match the force field parameters suitable for particular molecule at either end of the perturbation (λ 0.0 or λ 1.0). As two molecules often differ not only in their force field terms but also their geometry, it is often necessary to modify the internal coordinates as well. This is relatively easy In simple cases (morphing a methyl group into a hydrogen group) but for larger, complex, perturbations this is often cumbersome if not impossible. In the dual topology method no geometry variations are attempted. However, the interaction energy of a pair of solutes with their surroundings (solvent, protein, other solutes), is gradually turned on or off with the coupling parameter.

$$U(\lambda) = U_0 + \lambda U(S_2) + (1 - \lambda)U(S_1) \tag{4.16}$$

Equation (4.16) thus shows that at any given value of λ, the total energy of the system consists in a term U_0 that is independent of the perturbation and a term $U(S_2)$ and $U(S_1)$ which is a function of the intermolecular energies of the pair of solutes for which a free energy change is to be calculated.

A dual topology setup is simpler and more generally applicable than a single topology setup. However dual topology approaches suffer from a number of technical difficulties which are mainly related to the fact that if a solute does not have any intermolecular interaction with its surroundings, it can drift anywhere in the simulation box. This usually causes the free energy difference to converge very very slowly (in practice not at all). To overcome these difficulties, the dual topology technique implemented in ProtoMS constrains a pair of solutes to stay together by the introduction of dummy bond between the center of geometry of the two solutes. As this does not prove to be sufficient to avoid convergence issues, a soft-core non bonded energy function is also implemented. In essence, the function that computes the intermolecular energy of the solutes is modified such that when a solute is not fully interacting with its surroundings, it's Lennard-Jones and couloumbic energies are softened such that atomic overlaps do not result in very large, positive, energies. The solute is effectively 'softer'. There are three soft-core versions implemented in ProtoMS.

The original implementation in ProtoMS for a solute that is being turned off is described by equation (4.17).

$$U_{nonbonded,\lambda} = (1-\lambda)4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}^{12}}{(\lambda\delta\sigma_{ij}+r_{ij}^2)^6}\right)-\left(\frac{\sigma_{ij}^{6}}{(\lambda\delta\sigma_{iJ}+r_{ij}^2)^3}\right)\right]+\frac{(1-\lambda)^n q_i q_j}{4\pi\epsilon_0\sqrt{(\lambda+r_{ij}^2)}} \tag{4.17}$$

where the parameters n and δ control the softness of the Coulombic and Lennard-Jones interactions respectively.

An alternative that has been useful in some applications is described by equation (4.18)

$$U_{nonbonded,\lambda} = (1-\lambda)4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}^{12}}{(\lambda\delta\sigma_{ij}^6+r_{ij}^6)^2}\right)-\left(\frac{\sigma_{ij}^{6}}{\lambda\delta\sigma_{iJ}^6+r_{ij}^6}\right)\right]+\frac{(1-\lambda)^n q_i q_j}{4\pi\epsilon_0\left[\lambda\delta_c+r_{ij}^6\right]^{1/6}} \tag{4.18}$$

with an additional softness parameter δ_c for the Coulombic interactions.

Third, the soft-core implementation in the latest version of the Amber package is available and is described by equation (4.19)

$$U_{nonbonded,\lambda} = (1-\lambda)4\epsilon_{ij}\left[\left(\frac{\sigma_{ij}^{12}}{(\lambda\delta\sigma_{ij}^6+r_{ij}^6)^2}\right)-\left(\frac{\sigma_{ij}^{6}}{\lambda\delta\sigma_{iJ}^6+r_{ij}^6}\right)\right]+\frac{(1-\lambda)^n q_i q_j}{4\pi\epsilon_0\sqrt{(\lambda\delta_c+r_{ij}^2)}} \tag{4.19}$$

4.4 Generic Moves

ProtoMS conducts a simulation by performing a sequence of moves on the system. The following moves are currently implemented

- **Residue moves** Standard Monte Carlo (MC) moves on protein residues.
- **Solute moves** Standard MC moves on solute molecules.
- **Solvent moves** Standard MC moves on solvent molecule.
- **Volume moves** Monte Carlo moves that change the volume of the system. These are used to run constant pressure simulations.
- **GCSolute moves** Standard MC moves on GCsolute molecules.
- **Insertion moves** MC moves which selects a GCsolute with a θ value of 0 and turns it to 1
- **Deletion moves** MC moves which selects a GCsolute with a θ value of 1 and turns it to 0
- **Theta moves** MC moves which sample the value of θ on a GCsolute molecule
- **Sample moves** MC moves which sample the value of θ on a GCsolute molecule whilst applying a biasing potential λ-moves Monte Carlo moves that change λ. These may be used to perform umbrella sampling free energy simulations.
- **Dual potential moves** Works only with implicit solvent simulations. Allows to sample rapidly configurations with a crude potential but correct for errors with a specific acceptance test.

Residue Moves A residue move is a Monte Carlo move on a single protein residue. Obviously, for a residue move to be be performed, at least one protein that has flexible residues must be loaded. Each residue move comprises the following steps

1. A protein is picked randomly from the set of proteins that have flexible residues. Note that each protein is weighted equally, so each protein has an equal chance of being chosen, regardless of how many flexible residues it contains. This behaviour is likely to change in future versions of the code, as ideally the probability of choosing to move a protein should be proportional to the number of flexible residues.

2. One of the flexible residues within the protein is chosen randomly from the set of all flexible residues in the protein. Again, there is no weighting of residues, so each flexible residue has an even chance of being chosen, despite the size of each residue.
3. If the backbone of this residue is flexible, then a random number between 1 and 3 is generated. If the random number is equal to 1, then only a backbone move on the residue will be attempted. If the random number is equal to 2 then only a sidechain move will be attempted, where all of the flexible internals of the residue are moved. If the random number is equal to 3 then a backbone and sidechain move are attempted simultaneously. If the backbone of this residue is fixed, then only a sidechain move is attempted.
4. The change in energy that results from this move is evaluated, and then tested according to the Metropolis criterion to decide whether or not to accept the move.
5. If the move is accepted, then the new configuration of the residue is saved. If the move was rejected then the original configuration of the residue is restored.

You can change the flexibility of any residue in any protein by using the fixbackbone and fixresidues commands described in section *Miscellaneous*. All residues of all proteins are flexible by default, and have flexible backbones. Note that the backbone move is still experimental and not thouroughly tested. I recommend that you fix the backbone of all residues for production simulations. You control the maximum amounts that the residue moves via the residue template (see *Templates*). The actual amount that a residue moves by will be based on random values generated within the limits of the maximum amounts set in the residue template, e.g. if the maximum change of an angle was 5.0° , then the angle will be changed by a random value generated evenly between -5.0° and $+5.0^\circ$.

Solute Moves A solute move is a Monte Carlo move on a single solute molecule. Obviously, for a solute move to be performed, at least one solute molecule must be loaded. Each solute move comprises the following steps

1. A solute is picked randomly from the set of loaded solutes. Each solute is weighted equally, regardless of its size or numbers of degrees of freedom.
2. One of the residues is chosen at random within the solute. Again, each residue is weighted equally, regard- less of its size.
3. All of the flexible internals of this residue are changed, and the whole solute molecule is randomly translated, and rotated around its center of geometry.
4. The change in energy associated with this move is evaluated and then tested via the Metropolis criterion to decide whether or not to accept the move.
5. If the move is accepted then the new configuration of the solute is saved. If the move was rejected then the original configuration is restored. You can control the maximum amounts that the solute moves via the solute template (see *Templates*).

Solvent Moves

A solvent move is a Monte Carlo move on a single solvent molecule. Obviously, for a solvent move to be performed, at least one solvent molecule must be loaded. Each solvent move comprises the following steps

1. A solvent molecule is randomly chosen from the set of loaded solvent molecules. If preferential sampling is turned on (see *Simulation parameters*), then the solvent molecules closest to the preferred solute have a relatively higher weight, so will be more likely to be chosen. If preferential sampling is off, then each solvent is weighted equally, regardless of its relative size or proximity to a solute.
2. The solvent molecule is randomly translated and rotated around its center of geometry.
3. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion if preferential sampling was turned off, or via a biased Monte Carlo test if preferential sampling were turned on.
4. If the move was accepted then the new solvent configuration is saved, otherwise the original configuration is restored.

You can control the maximum amounts that the solvent is translated and rotated by by editing its solvent template (see *Templates*).

Volume Moves

A volume move is a Monte Carlo move that changes the volume of the system. This is needed to be able to perform Monte Carlo simulations at constant pressure (i.e. using the NPT ensemble). For a volume move to be performed you need to have loaded a box of solvent molecules, and be running using periodic boundary conditions. A volume move is comprised of the following steps

1. A random change in volume is chosen within the range set via the maxvolchange command (see *Simulation parameters*).
2. The volume of the system is changed by this amount by scaling all of the coordinates evenly from the center of the simulation box.
3. The change in energy associated with this change in volume is evaluated and used to decide whether or not to accept this move via the constant pressure Monte Carlo test, for the system pressure set via the pressure command (see *Simulation parameters*).
4. If the move is accepted then the new system configuration is saved, otherwise the original system configuration is restored.

GCsolute Moves

A GCsolute move is a Monte Carlo move on a single Gcsolute molecule. Each GCsolute move comprises the following steps

1. A GCsolute molecule is randomly chosen from the set of loaded GCsolute molecules. The value of θ is examined; if it is set to 0 then another is chosen until the examined θ value is 1. If no GCsolutes with $\theta = 1$ are available then the GCsolute move is counted as rejected.
2. The GCsolute molecule is randomly translated and rotated around its center of geometry. If it attempts to leave the confines of its predefined cubic region then it experiences a huge energetic penalty, ensuring that the Metropolis move is rejected.
3. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion.
4. If the move was accepted then the new GCsolute configuration is saved, otherwise the original configuration is restored.

You can control the maximum amounts that the GCsolute is translated and rotated by by editing its template (see *Templates*). If performing a jaws simulation then a GCsolute is chosen at random in 1. without examining its θ value.

Insertion Moves

An insertion move is a Monte Carlo move on a single GCsolute molecule, whereby the θ value of a GCsolute is turned from 0 to 1. Each insertion move comprises the following steps;

1. A GCsolute molecule is randomly chosen from the set of loaded GCsolute molecules. The value of θ is examined; if it is set to 1 then another is chosen until the examined θ value is 0
2. The GCsolute molecule is given a random position and orientation within the GCMC region.
3. The value of θ for that GCsolute molecule is set to 1, and the new energy associated with this value of θ is calculated
4. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion.
4. If the move was accepted then the new value of θ for that GCsolute molecule is saved, otherwise the original value of 0 is restored.

Deletion Moves

A deletion move is a Monte Carlo move on a single GCsolute molecule, whereby the θ value of a GCsolute is turned from 1 to 0. Each deletion move comprises the following steps

1. A GCsolute molecule is randomly chosen from the set of loaded GCsolute molecules. The value of θ is examined; if it is set to 0 then another is chosen until the examined θ value is 1. If no GCsolutes with $\theta = 1$ are available then the deletion move is counted as rejected.
2. The value of θ for that GCsolute molecule is set to 0, and the new energy associated with this value of θ is calculated
3. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion.
4. If the move was accepted then the new value of θ for that GCsolute molecule is saved, otherwise the original value of 1 is restored.

Theta Moves

A theta move is a Monte Carlo move on a single GCsolute molecule, whereby the θ value of a GCsolute is sampled. Each theta move comprises the following steps

1. A GCsolute molecule is randomly chosen from the set of loaded GCsolute molecules
2. The value of θ for that GCsolute molecule is randomly changed, and the new energy associated with this value of θ is calculated
3. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion.
4. If the move was accepted then the new value of θ for that GCsolute molecule is saved, otherwise the original value of θ is restored.

Sample Moves

A sample move is a Monte Carlo move on a single GCsolute molecule, whereby the θ value of a GCsolute is sampled whilst applying a biasing potential, jbias. Each sample move comprises the following steps

1. A GCsolute molecule is randomly chosen from the set of loaded GCsolute molecules (typically only one GCsolute molecule is studied in a sample move)
2. The biasing potential is added onto the value of ieold for that molecule, based upon the volume of the restraint and the applied jbias
3. The value of θ for that GCsolute molecule is randomly changed, and the new energy associated with this value of θ is found
4. The biasing potential is added onto the value of ienew for that molecule, based upon the volume of the restraint and the applied jbias
5. The change in energy associated with this move is evaluated and used to decide whether or not to accept this move via the Metropolis criterion.
6. If the move was accepted then the new value of θ for that GCsolute molecule is saved, otherwise the original value of θ is restored.

Relative Move Probabilities

You can specify which moves should be run by passing arguments to the simulate and equilibrate commands (see *Running a Simulation*). You can use these commands to assign a weight to each type of move, e.g. 100 for solvent moves, 10 for protein moves, 1 for solute moves and 0 for volume move. The type of move chosen for each step of the simulation is generated randomly based on these set relative weights. These weights mean that on average, in 111 moves, 100 of these moves will be solvent moves, 10 of these moves will be protein moves, 1 of these moves will be

solute moves and none of the moves will be volume moves (e.g. no volume moves will be performed). Note that you need to perform some volume moves if you wish to sample from the NPT ensemble!

CHAPTER

FIVE

EXECUTING PROTOMS

ProtoMS is a simple program that may be used from the command line. Once you have compiled it you should find it in the top directory (it is called simply protoms3). If you run the program you should see that it prints out some information about the program and license, then it complains that nothing has been loaded so it closes down. The interface to ProtoMS has been designed to allow easy integration of ProtoMS with scripts, and to enable simple use from a command file. A ProtoMS input consists of a set of commands and values, e.g. the command `temperature` could have the value *25.0* . This would set the simulation temperature to 25° C. The input is passed to ProtoMS via a command file. The above command could thus be input by setting by placing the line

```
temperature 25.0
```

into a file and have ProtoMS read commands from that file. You specify the command file by passing it to ProtoMS on the command line, e.g.

```
protoms3 mycmdfile.txt
```

Note that ProtoMS is insensitive to whether commands, variables or contents of files are uppercase or lowercase, so you are free to mix and match capitals and small case wherever you want. The only exception to this is in the specification of filenames, where your operating system may care about case. As an example, depending on your operating system, ProtoMS may fail when the file containing the commands is named in upper case letters.

For replica exchange or ensemble type calculations, you have to execute ProtoMS through an appropriate MPI wrapper, e.g.

```
mpirun -np 16 protoms3 mycmdfile.txt
```

5.1 File output

If you run ProtoMS from the command line you should see that it prints out a lot of information to the screen (on Unix called standard output, STDOUT). If you look closely at the output you should see that each line of output is preceeded by a tag, such as 'HEADER' or 'INFO'. ProtoMS uses streams to output data, and these tags state which stream the line of data came from. Thus the information at the top of the output that gives the license and version details has been printed to the 'HEADER' stream, while the lines stating that ProtoMS is closing down because nothing has been loaded have gone to the 'FATAL' stream. ProtoMS uses the following streams

- **HEADER** Used to print the program header.
- **INFO** Used to print general information.
- **WARNING** Warnings are printed to this stream. ProtoMS will generally try to continue if it detects a problem, and will print out information about any errors to the WARNING stream. It is up to you to check the WARNING stream to ensure that your simulation is working correctly.

- **FATAL** If an error is so serious that ProtoMS is forced to shutdown then it will first try to tell you what the problem is by sending text to the FATAL stream.
- **RESTART** The restart file is written to the RESTART stream.
- **PDB** Any output PDB files are written to the PDB stream.
- **MOVE** Information about moves are printed to this stream, e.g. whether or not a move was accepted, and how much progress has been made during the simulation.
- **ENERGY** Information about the energy components for the moves are printed to this stream, e.g. the bond energy of solute 1, or the coulomb energy between protein 1 and the solvent.
- **RESULTS** The results of the simulation are written to the RESULTS stream. These include the free energy averages and energy component averages.
- **DETAIL** The DETAIL stream contains lots of additional detail about the setup of the simulation. This can be very verbose, as it includes complete detail of the connectivity of the system and the loaded forcefield. The DETAIL stream is useful when you are setting a simulation up, though should be turned off when you are running production.
- **SPENERGY** The SPENERGY stream is used to report the results of single point energy calculations.
- **ACCEPT** The ACCEPT stream is used to print information about the numbers of attempted and accepted moves.
- **RETI** The RETI stream is used to report the energies needed by the RETI free energy method.
- **DEBUG** The DEBUG stream is used by the developers to report debugging information during a ProtoMS run. This stream is only active if 'debug' is set to true.

These streams may be switched on or off, directed to STDOUT, directed to STDERR or directed to a file. You can do this by using the commands

```
streamSTREAM STDOUT

streamSTREAM STDERR

streamSTREAM off

streamSTREAM /path/to/file.txt
```

where *STREAM* is the name of the stream that you wish to direct (e.g. streamINFO). ProtoMS is insensitive to case, so you could use the command

```
streaminfo stdout
```

However, your operating system may be sensitive to case so you should ensure that you use the correct case for filenames.

You are free to direct multiple streams into a single file, or to turn undesired streams off. If a stream is output to STDOUT or STDERR then the name of the stream is prepended to the start of each line. The name is not attached if the stream is directed into a file. The WARNING and FATAL streams are special as unlike the other streams, these two cannot be turned off. These two streams will be directed to STDERR if they have not been directed elsewhere.

By default, the HEADER, INFO, MOVE and RESULTS streams are directed to STDOUT, the WARNING and FATAL streams are directed to STDERR, and the remaining streams are switched off. Bear this in mind if you think that you should be getting output and you are not - make sure that the stream that contains your output is directed to something!

The streamSTREAM command is used to specify the direction of the stream at the start of the simulation. It is possible to redirect streams while the simulation is running. This is slightly more complicated than then streamSTREAM command, and is described in section *Miscellaneous*.

By default ProtoMS overwrites the files specified by the streamSTREAM command. If you want to append to already exisiting files, for instance if you are restarting a simulation, you have to add the option

```
appendstreams on
```

This option will turn on append mode for all streams, except the **RESTART** stream that never will be appended.

5.2 Simulation parameters

There are many commands to set parameters that you can use to control your simulation. To make it easier to search for those relevant to your calculations, these will be divided in several subsections.

In the subsections below, unless otherwise specified:

- `locical` stands for *true* or *false*, *yes* or *no*, *on* or *off* (depending on your personal preference)
- `integer` or `int` stands for any integer number
- `float` stands for any floating point number
- `string` stands for a string of characters

5.2.1 Parameters for developers

```
debug logical
```

This turns on or off debugging output that may be useful for ProtoMS developers. By default `debug` is *off*.

```
testenergy logical
```

This is used to set whether or not to turn on testing of energies. This is useful if you are developing ProtoMS. By default `testenergy` is *off*.

5.2.2 General parameters

```
prettyprint logical
```

Turn on or off pretty printing. With pretty printing turned on, you will see nice starry boxes drawn highlighting certain parts of the output. By default, `prettyprint` is *on*.

```
dryrun logical
```

Whether or not to perform a dry run of the simulation. If this is true then all of the files will be loaded up and your commands parsed. If there are any problems then these will be reported in the WARNING stream. No actual simulation will be run, though any files that would be created may be created. While this option is very useful for testing your commands, it is not perfect and cannot check everything. I thus recommend that you also perform a short version of your simulation before you commit yourself to full production. By default `dryrun` is off.

```
ranseed integer
```

where `integer` is any positive integer. This command is used to set the random number seed to be used by the random number generator. The random number seed can be any positive integer, and you will want to specify a seed if you wish to run reproducable simulations. If you do not specify a random number seed then a seed is generated based on the time and date that the simulation started.

```
temperature float
```

Use this command to specify the simulation temperature in *Celsius*. By default temperature is 25.0 C.

```
pdbparam logical
```

Whether or not to automatically detect and use, in the simulation, any chunks which might be included in the input PDB files after REMARK. It is most commonly used to include the fixresidues and fixbackbone commands often found at the beginning of a protein scoop. Any chunks included in pdb files will be applied before any other chunk. By default *pdbparam* is on.

```
cutoff float
```

where `float` is any positive number. This command is used to set the size of the non-bonded cutoff, in Angstroms, used to truncate the intermolecular non-bonded potentials (see eq (4.1)). By default the non-bonded cutoff is 15A.

```
feather float
```

To prevent an abrupt cutoff, the non-bonded energy is scaled quadratically down to zero over the last part of the cutoff (see eq (4.1)). The feather command sets the distance over which this scaling occurs, e.g.

```
feather 1.3
```

sets this feathering to occur over the last 1.3A. The default value of the feather is 0.5A.

```
cuttype string
```

where `string` is either *residue* or *molecule*. This specifies the type of non-bonded cutting to use; either residue, where the cutoff is between protein residues, solute molecules and solvent molecules, or molecule, where the cutoff is between protein molecules, solutes molecules and solvent molecules. By default the `cuttype` is *residue*.

```
pressure float
```

This command sets the pressure of the system in atmospheres. By setting the pressure to a non-zero value you will be able to perform a simulation in the NPT isothermal-isobaric ensemble. Note that you need to perform volume moves (see *Generic Moves*) to be able to run in the NPT ensemble. By default the pressure is equal to zero, and thus a NPT simulation is not performed.

```
maxvolchange float
```

This command sets the maximum change in volume for a volume move in cubic Angstroms. This command only has meaning if an NPT simulation is being performed. By default `maxvolchange` is equal to the number of solvent molecules divided by ten.

```
prefsampling integer
```

This command is used to turn on preferential sampling of the solvent, and to specify which solute is used to define the center of the preferential sampling sphere. The command

```
prefsampling 1
```

means that the solvents closest to solute 1 will be moved more frequently than those furthest from solute 1. An optional parameter may be used to change the influence of the sphere, e.g.

```
prefsampling 1 100.0
```

will specify a preferential sampling sphere centered on solute 1, with a parameter of 100.0. The larger the parameter, the more highly focussed the influence of the sphere around the closest solvent molecules. By default the parameter is 200.0, and preferential sampling is turned off.

```
boundary none
```

This turns off any boundary conditions, i.e. the simulation will be performed in vacuum.

```
boundary periodic dimx dimy dimz
```

This turns on periodic boundaries, using a orthorhombic box centered on the origin, with dimensions `dimx` A by `dimy` A by `dimz` A. Note that these dimensions may be modified by any loaded solvent file

```
boundary periodic ox oy oz tx ty tz
```

This turns on periodic boundaries using an orthorhombic box with the bottom-left-back corner at coordinates (`ox` , `oy` , `oz`) A and the top-right-front corner at (`tx` , `ty` , `tz`) A. Note that these dimensions may be modified by any loaded solvent file.

```
boundary cap ox oy oz rad k
```

This turns on solvent cap boundary conditions. Protein and solute molecules will experience no boundary conditions, while solvent molecules will be restrained within a spherical region of radius rad A, centered at coordinates (`ox` , `oy` , `oz`) A. A half-harmonic restraint with force constant `k` kcal.mol-1.A-2 is added to the solvent energy if it moves outside of this sphere.

```
boundary solvent
```

This sets the boundary conditions to whatever is set by the loaded solvent files. If no solvent files are loaded then no boundary conditions are used. This is the default option, and the method of setting boundary conditions via a solvent file is described in section *Solvent File*

5.2.3 GBSA parameters

```
surface quality 3 probe 1.4
```

This command will cause surface area calculations to be performed during the simulation. `quality` can be set to 1,2,3,4 and will result in increasingly precise surface area calculations. For typical simulations, 3 should be fine and 2 will not give a huge error. `probe` is the radius of the probe and should be set to 1.4 if you want to calculate the solvent accessible surface area of water, but can be set to 0 if you want to calculate the van der waals surface area of a molecule.

```
born cut 20 threshold 0.005 proteins
```

This command will enable Generalised Born energy calculations. Thus to run a full GBSA simulation you should use both the surface and born keywords. cut controls the cutoff distance for the computation of the Born radii. If you work with a medium sized protein scoop of circa 100-150 residues, 20 should be fine but you may want a larger value for simulations of large proteins. threshold controls the number of pairwise terms that are not updated when the effective Born radii must be calculated by the Pairwise descreening approximation. The default value 0.005 appear to be a good tradeoff. Increasing it will make the simulation faster but less accurate. proteins activates the rescaling of the Born radii to compensate for systematic errors of the Pairwise Descreening Approximation in large biomolecules. It should be used only when simulating proteins and then its effectiveness has not been yet convincingly demonstrated.

WARNING: These commands are considered to be deprecated. This means that they are not developed any more and have not been tested extensively with newer features. Dump commands are supported with the *simulate* command and

one can do simple MC sampling with GB. However, it is not sure that free energy or the replica-exchange commands work satisfactorily.

5.2.4 Temperature replica-exchange parameters

ProtoMS can perform temperature replica-exchange simulations.

```
temperaturere integer float float float
```

is the command to set a replica exchange simulation between the different temperatures given as floats, where `float` is any positive float, and temperatures are given en Celsius. In principle, any desired number of temperature values can be used, and the simulation will require to be runned in as many cores as temperature values are provided. The integer value stands for the frequency at which the exchange between the different temperature values is attempted. Please, note that this value should be a multiple of the frequency of printing output when the dump commands are used (see *Frequent output generation*). If no exchange is desired, the frequency of exchange can simply be set to the total number of moves of the simulation.

As an example:

```
temperaturere 20 25.0 30.0 35.0
```

corresponds to a simulation which will run at three different temperature windows in parallel, and will attempt swaps between the conformations of different temperature windows each 20 moves.

The temperature replica-exchange command can be used in conjuction with the `lambdare` command, see below, to add temperature ladders to different values of λ.

```
solutetempering 25.0 bndang 3 dih 1 lj 3 coul 1 solu 2 prot 2 solv 2
```

Turns on replica-exchange with solute tempering (REST). It only works if you have specified temperature replica-exchange (see *Temperature replica-exchange parameters*). In this type of simulation the system is simulated at 25.0 Celsius, or the temperature set with this command, and the temperatures set with the `temperaturere` command are used to scale the solute energies. The level of scaling for the different energy components can be set with the rest of the options; `bndang` controls the internal bond-angle energy terms, `dih` the internal dihedral energy term, `lj` the internal van der Waals energy, `coul` the internal Coulomb energy, `solu` the interaction with other solutes, `prot` the interaction with the protein and `solv` the interaction with solvent molecules. Each argument can be either 1, 2 or 3. If the argument is 1, the energy is scaled with β_i/β_0, where β_i is the effective inverse temperature of the replica (set with the `temperaturre` command) and β_0 is the inverse simulation temperature (set with this command). If the argument is 2, the energy is scaled with $(\beta_i + \beta_0)/2\beta_0$ and if the argument 3 the energy is unscaled.

By default ProtoMS will use different random seeds for each replica in a replica exchange simulation. Setting sameseeds to true will prevent this and all replicas will use the random seed provided in the command file. This is primarily available for backwards compatibility:

```
sameseeds logical
```

The value of sameseeds is false by default.

5.2.5 Free energy calculation parameters

To be able to run a single simulation for a given lambda value, you will need to use the following parameters:

```
lambda float
```

where `float` is a number between 0.0 and 1.0. Specify the value of λ. If a single value is given then that is used for λ. If three values are given then these are used for λ, and λ in the forwards and backwards windows, e.g.

```
lambda 0.5 0.6 0.4
```

would set λ for the reference state to 0.5, λ for the forwards perturbed state to 0.6, and λ for the backwards perturbed state to 0.4. By default all values of λ are 0.0.

To run several at several values of λ in parallel and hence perform your full perturbation at once with ProtoMS, you will need the commands shown below. Running your free energy calculation in this manner, you will be able to attempt exchanges between the configurations of your system at the different lambdas, increasing the probability of convergence.

```
lambdare integer float float float
```

is the command to set a replica exchange calculation between the different λ given as floats, where `float` is a number between 0.0 and 1.0. In principle, any desired number of λ values can be used, and the simulation will require to be runned in as many cores as λ values are provided. The integer value stands for the frequency at which the exchange between the different λ values is attempted. Please, note that this value should be a multiple of the frequency of printing output when the dump commands are used (see *Frequent output generation*). If no exchange is desired, the frequency of exchange can simply be set to the total number of moves of the simulation.

As an example:

```
lambdare 20 0.000 0.333 0.667 1.000
```

corresponds to a simulation which will run at four different λ windows in parallel, and will attempt swaps between the conformations of different λ windows each 20 moves.

With temperature replica-exchange

```
temperatureladder lambda float float
```

is one of the commands required to proceed with a simulation including both temperature and λ replica exchange, where `float` is each of the λ values where a temperature ladder is desired. All λ values must be among those included after the `lambdare` keyword. In principle, the number of temperature ladders can be as high as the number of λ windows.

As an example:

```
temperatureladder lambda 0.00 1.00
```

corresponds to a simulation which runs the λ windowns 0.00 and 1.00 at all temperatures included after the `temperaturere` keyword, as far as the corresponding `lambdaladder` command line is set accordingly.

```
lambdaladder temperature float float
```

is one of the commands required to proceed with a simulation including both temperature and λ replica exchange, where `float` is each of the temperature values where a λ ladder is to be placed. All temperature values must be among those included after the `temperaturere` keyword. In principle, the number of lambda ladders can be as high as the number of temperatures in temperaturere. The number of cores must be calculated based on the number of λ ladders and temperature ladders, as well as λ and temperature values per ladder, takind into account the cores shared by each λ ladder with each temperature ladder.

As an example:

```
lambdaladder temperature 25.0 35.0
```

corresponds to a simulation which runs all λ windowns at temperatures 25.0 and 35.0, as far as the corresponding `temperatureladder` command line is set accordingly.

All replica-exchange commands together:

```
lambdare 20 0.000 0.333 0.667 1.000
temperaturere 20 25.0 30.0 35.0
temperatureladder lambda 0.00 1.00
lambdaladder temperature 25.0 35.0
```

correspond to a simulation where λ windows 0.000 0.333 0.667 1.000 are simulated at 25.0 and 35.0 Celsius, while at temperature 30.0, only λ windows 0.000 and 1.000 will be simulated.

Other free energy commands

```
dlambda float
```

where `float` is a number between 0.0 and 1.0 (often of the order of 0.001). This command sets the gradient for a free energy calculation. It is required for thermodynamic integration (TI) to be applied on the simulation results.

```
printfe string
```

where `string` should be either `off`, `bar` or `mbar`. Whether to print the free energy estimates required to proceed with BAR or MBAR calculations. Take into acount that this estimates will take some time. Your simulations may run faster when this option is set to off (default).

In case dual topology is desired, whether it is for a single or multiple λ simulation, the following parameters must be used:

```
dualtopologyint integer1 integer2 synctrans syncrot
```

This turns on the dual topology method of calculating relative free energies, where *int1* is the perturbed solute at λ = 0.0 and *in2* is the solute at λ = 1.0 . If `synctrans` is set, the rigid body translations of the two solutes will be synchronised. If `syncrot` is set, the rigid body rotations of the two solutes will also be synchronised.

```
softcoreint solute integer
```

This causes the intermolecular energy of solute integer to be softened. Alternatively, you can write `all` instead of the solute index and all solutes will have their non bonded energy softened. The softcore is only supported for solutes. May alternatively be used as below.

```
softcoreint solute integer atoms int1 int2 int3...
```

The atoms keyword indicates that the interactions of only a subset of atoms within a solute should have their interactions softened. Whilst int1, int2, e.t.c. provide indices specifying atoms based on their ordering within the solute template. In principle, superior numerical convergence can be achieved by applying softcores to only the smallest subset of atoms required to prevent singularities.

```
softcoreparams coul 1 delta 1.5 gb 0 old
```

This causes the solutes non bonded energy to be softened with a parameter n set to 1 and δ set to 1.5. (see eq (4.17)). The old keyword selects the original soft-core implementation and can be omitted. If conducting a GBSA simulation, this also causes the GB energy to be softened as well. It is recommended to use the same parameter for the Coulombic and Generalised Born energy. The values listed here, seem to work well for a number of relative binding free energy calculations but actual optimum values of these parameters will depend on your system.

```
softcoreparams coul 1 delta 0.2 deltacoul 2.0 soft66
```

This causes the solutes non bonded energy to be softened with a parameter *n* set to 1, δ set to 0.2 and δ_c set to 2.0. (see eq (4.18)). The soft66 keyword selects the second soft-core implementation, eq (4.18) .

```
softcoreparams coul 1 delta 0.5 deltacoul 12.0 amber
```

This causes the solutes non bonded energy to be softened with a parameter *n* set to 1, δ set to 0.5 and δ_c set to 12.0. (see eq (4.19)). The amber keyword selects the third soft-core implementation, eq (4.19). The values listed here are the default values in the Amber package.

5.2.6 GCMC and JAWS parameters

```
gcmc 0
```

This command tells ProtoMS that it is to perform a GCMC simulation, and that the starting value of Θ all of the GCsolutes is 0.

```
potential float
```

This command will set a *B*-value of `float` (i.e. -8) for moves in the Grand Canonical Ensemble. The value of *B* can be related to the excess chemical by the following equation:

$$B = \frac{\mu'}{k_B T} + \ln \bar{n} \tag{5.1}$$

In the equation, $\bar{n}$ is the number density of the GCsolute multiplied by the simulation subvolume.

```
multigcmc integer float float float
```

is the command to run multiple gcmc simulations in parallel with replica exchange between different B values. The integer value sets how often replica exchange moves are attempted, this should be some multiple of how often results files are written. Each `float` is the B value for each replica. In principle, the number of B values is not restricted. The simulation will need to be submited to run in parallel with as many cores as B values. As above the random seed used by each replica can be influenced by the sameseeds command.

```
originx float
```

This command will set the X origin of the defined GCsolute sampling subvolume to be the specified `float`

```
originy float
```

This command will set the Y origin of the defined GCsolute sampling subvolume to be the specified `float`

```
originz float
```

This command will set the Z origin of the defined GCsolute sampling subvolume to be the specified `float`

```
x float
```

This command will set the distance along the X coordinate from originx to be the specified `float`

```
y float
```

This command will set the distance along the Y coordinate from originy to be the specified `float`

```
z float
```

This command will set the distance along the Z coordinate from originz to be the specified `float`

Alternatively to the origin, the position of the box may be set using its center:

```
centerx float
```

This command will set the X center of the defined GCsolute sampling subvolume to be the specified `float`

```
centery float
```

This command will set the Y center of the defined GCsolute sampling subvolume to be the specified `float`

```
centerz float
```

This command will set the Z center of the defined GCsolute sampling subvolume to be 9

A different, equally valid expression for the distance or length of the box is the keyword *len?*:

```
lenx float
```

This command will set the distance along the X coordinate from originx to be the specified `float`

```
leny float
```

This command will set the distance along the Y coordinate from originy to be the specified `float`

```
lenz float
```

This command will set the distance along the Z coordinate from originz to be the specified `float`

```
jaws1 0
```

This command tells ProtoMS that it is to perform a JAWS stage one simulation, and that the starting value of θ all of the GCsolutes is 0.

```
thres 0.95
```

This command will set the θ threshold for defining whether a molecule is *on* in the first stage of the JAWS method to be 0.95 (default)

Note here that, in order to run a JAWS stage 1 calculation, you will also need to include softcores. The parameters to do this can be found among the *Free energy calculation parameters*.

```
jaws2 1
```

This command tells ProtoMS that it is to perform a JAWS stage two simulation, and that the starting value of θ all of the GCsolutes is 1.

```
jbias float
```

This command will set the value of the biasing potential in the second stage of the JAWS algorithm to be `float`, in kcal/mol (i.e. 14).

5.3 Specifying input files

As well as controlling the simulation, commands are also used to specify the names of the input files that describe the system and forcefield for the simulation. These input files are specified using the following commands

```
proteinN filename
```

Specifies the name of the Nth protein file, e.g.

```
protein1 protein.pdb
```

specifies that protein 1 should be loaded from the file protein.pdb. Note that proteins must be numbered sequentially from 1 to MAXPROTEINS. The format of a protein file is described in *Protein File*.

```
soluteN filename
```

specifies the name of the Nth solute file. Note that the solutes must be numbered sequentially from 1 to MAXSOLUTES. The format of a solute file is described in section *Solute File*.

```
solventN filename
```

specifies the name of the Nth solvent file. Unlike the protein and solute files, the solvent file may contain multiple solvent molecules, though the total number of solvent molecules cannot exceed MAXSOLVENTS. The format of a solvent file is described in section *Solvent File*.

```
grandN filename
```

specifies the name of the Nth GCsolute file. Unlike the protein and solute files, the GCsolute file may contain multiple GCsolute molecules, though the total number of GCsolute molecules cannot exceed MAXSOLVENTS. The format of a GCsolute file is described in section *GCsolute File*.

```
parfile filename
```

Specify the name of a forcefield parameter file. You can specify as many parameter files as you wish. The list of parameter files is read from top to bottom, such that if any paramater files contain contradictory information, the last parameters read by ProtoMS are used. The format of the parameter file is described in section *Parameter / Forcefield Files* .

5.4 Running a Simulation

There are two main keywords related to running a simulation. These are *chunk* and *dump*. All individual actions (commands which ProtoMS should perform only as it is prompted to do so) are handled with *chunk* lines. Actions which ProtoMS should perform with a certain frequency *while* the simulation is running, are handled with *dump* lines. We can start by talking about chunks.

A simulation can be run as a sequence of chunks. Different things may be accomplished in each chunk, e.g. running some steps of equilibration, printing the protein coordinates to a PDB or redirecting a stream to a new file. Chunks may be mixed and matched, and you can run as many chunks as you desire within a single simulation. You specify a chunk using the command

```
chunk command
```

Chunks are executed in the order they appear in the command file.

5.4.1 Equilibration and Production

The meat of a simulation is equilibration and production. In ProtoMS equilibration is defined as sampling without the collection of free energy or energy averages, while production is sampling with the collection of free energy and energy averages. Equilibration and production are specified using the equilibrate and simulate chunks, e.g.

```
chunk equilibrate 50
```

performs 50 steps of equilibration.

```
chunk simulate 1000
```

performs 1000 steps of production.

Additional options may be passed to these two chunks to control the probability of different types of move and the frequency of printing out move and energy details to the MOVE and ENERGY streams. These options are

```
printmove=N
```

Print move and energy information every N moves.

```
protein=N
```

Set the relative probability of protein moves to N.

```
solute=N
```

Set the relative probability of solute moves to N.

```
solvent=N
```

Set the relative probability of solvent moves to N.

```
gcsolute=N
```

Set the relative probability of gcsolute moves to N.

```
insertion=N
```

Set the relative probability of insertion moves to N.

```
deletion=N
```

Set the relative probability of deletion moves to N.

```
theta=N
```

Set the relative probability of GCsolute theta moves to N.

```
sample=N
```

Set the relative probability of GCsolute sample moves to N.

```
volume=N
```

Set the relative probability of volume moves to N.

```
newprob
```

Reset relative move probabilities to zero.

Note that succeeding equilibration or production chunks inherit the move probabilities and printing frequency of preceeding simulation or equilibration chunks. I thus recommend that you use the newprob option to reset the move probabilities for each equilibration or production chunk you run.

The following examples illustrate the use of these options;

```
chunk newprob equilibrate 500 printmove=10 protein=1 solvent=1000
```

Perform 500 steps of equilibration, printing move and energy information every 10 moves, making on average 1 protein move for every 1000 solvent moves (and performing no other types of move).

```
chunk equilibrate 100 solute=500
```

Perform 100 steps of equilibration. Because this chunk will inherit from the previous chunk, the move and energy information will still be printed every 10 moves, and still, on average 1 protein move will be made every 1000 solvent moves. However this line has added that on average 500 solute moves should be made for every 1000 solvent moves, thus the probability of a protein move is now 1 in 1501, the probablity of a solute move is 500 in 1501, and the probability of a solvent move is 1000 in 1501.

```
chunk simulate 500 printmove=1 newprob volume=1 solvent=300
```

Now perform 500 steps of production, printing move and energy information every move, performing no protein moves, and 1 volume move for every 300 solvent moves.

A couple of *simulate-like* commands are specifically related to GBSA simulations.

```
chunk splitgbsasimulate 100 10 solute=1 protein=9
```

The above command should only be used if you are doing an implicit solvent simulation (e.g, you turned on the surface and born keywords). This will cause to run 10 moves with a crude GBSA potential and then perform an acceptance test based on the difference of energies between the crude GBSA potential and the GBSA potential you set with the cutoff, born and surface keywords. This will be repeated 100 times. Here the move probabilities were set to 1 and 9 for solute and protein, but could be other figures. After this keyword has been used it is advised to use the following keyword.

WARNING: this command is deprecated. For instance, it does not support the dump commands

```
chunk resetgb
```

This will cause the total energy of the system to be calculated fully and the Born radii to be correctly updated. Periodic usage of this command, along with the previous one, avoids drifts in the total energy of the system.

5.4.2 Results and Restarts

As well as controlling the sampling, you can also control the collection and output of results using simulation chunks, and the reading and writing of restart files.

```
chunk results reset
```

Reset all averages to zero and start collection of results from scratch.

```
chunk results write
```

Write out the energy and free energy averages to the RESULTS stream. It is probably a good idea to do this a some point before the end of the simulation.

```
chunk results write myfile.txt
```

Does the same as above, but redirects the RESULTS stream to myfile.txt before the results are written.

```
chunk results writeinst myfile.txt
```

Does the same as *write* but instead writes instantaneous energies (the energies of the current snapshot) rather than average energies. This can be useful for some analyses.

Note that all the *chunk averages* lines above are equally valid, if *results* is written instead of *averages*:

```
chunk results write myfile.txt
```

```
chunk restart write
```

Write a restart file for the current configuration to the RESTART stream.

```
chunk restart write myfile.txt
```

Does the same as above, but redirects the RESTART stream to myfile.txt before the restart file is written.

```
chunk restart read myfile.txt
```

Read in a restart file from the file myfile.txt. The expected behaviour of ProtoMS varies depending on the restart file provided. If the restart file was written from a simulation that DID NOT complete it's target number of Monte Carlo steps then reading the restart file will cause ProtoMS to attempt to complete the remaining number of required simulation steps. This is ideal for completing simulations that have been inadvertently stopped before completion. This will only work for restart files written by ProtoMS 3.4. To force the simulation step count to start from zero you can edit the restart file and change the value of the entry on the first line to a T. If the restart file was written from a simulation that DID complete it's target number of Monte Carlo steps then the restart data will simply override the data from the pdb inputs and the simulation step count will start from zero.

5.4.3 PDB Output

You can use a simulation chunk to output a PDB of the current configuration. The output can be tailored to include only the parts of the system that you are interested in. This is useful if you are trying to conserve disk usage. You can output PDBs using the 'pdb' chunk

```
chunk pdb all
```

Output a PDB of all proteins and solutes to the PDB stream

```
chunk pdb protein=all
```

Output a PDB of all proteins to the PDB stream

```
chunk pdb protein=2
```

Output a PDB of protein 2 to the PDB stream

```
chunk pdb solute=all
```

Output a PDB of all solutes to the PDB stream

```
chunk pdb solute=1
```

Output a PDB of solute 1 to the PDB stream

The output PDB can be controlled via additional commands added to the above lines, e.g.:

```
chunk pdb all solvent=all
```

Output the PDB including all solvent molecules.

```
chunk pdb solute=1 solvent=5.0
```

Output a PDB including all solvent molecules within 5.0A of whatever else is printed - in this case solute 1.

```
chunk pdb protein=1 showdummies
```

Output a PDB that also includes dummy atoms.

```
chunk pdb solute=all showhidden
```

Output a PDB that also includes hidden solute molecules (solutes that are used to perform geometry perturbations).

```
chunk pdb all file=myfile.txt
```

Redirect the PDB stream to myfile.txt then print the PDB.

```
chunk pdb all solvent=all standard
```

Output a PDB that have a more standard format than normal, such that it can be viewed and interpreted correctly in most programs.

5.4.4 Restraints

ProtoMS supports a number of restraining potentials which can be used to modify the potential energy function and bias the simulation towards particular configurations. To use a restraint in ProtoMS you must first assign an id number to a particular atom or set of atoms, using the following command

```
chunk id add int1 type int2 atname resname|resnumber
```

where *int1* is the index numberr for this id. So if this if the first id you create you may want to use the number 1. type can be SOLUTE or SOLVENT or PROTEIN depending on where the atom you want to tagg is. *atname* is the name of the atom (e.g CA), *resname* is the name of the residue the atom is in if you are dealing with a SOLUTE or SOLVENT. However if it the atom is in a protein, then you must use the PDB residue number. Once you have specified a few ids, you can create restraints using these ids and the following command

```
chunk restraint add id1[-id2-id3-id4] type1 type2 [other parameters]
```

where *id1* to *id4* designate up to four ids. *type1* designate the type of the restraint. It can be either *cartesian*,'bond' or *dihedral*. In the first case the restraint is applied in cartesian coordinates and will apply to only one atom (*id1*). In the second case, it is applied in internal coordinates, and will apply to only two atoms (*id1-id2*). In the last case it is applied to four atoms (*id1-id2-id3-id4*) and in internal coordinates. *type2* designate the functional form of the restraint. It can

be *harmonic* or *flatbottom*. Each functional form requires additional parameters. The following options are currently possible:

```
chunk restraint add id1 cartesian harmonic xrest yrest zrest krest [lambda]
```

For a cartesian harmonic restraint you need to specify the coordinates of the anchoring point and the value of the force constant. You may optionally give the keyword lambda at the end of the chunk to scale the energy of the restraint by the value of lambda. This means that when lambda=1 the restraint is on and when lambda=0 the restraint is off. This is useful to calculate the free energy change associated with the introduction of a restraint, although this procedure should not be carried out unless for the bound phase:

```
chunk restraint add id1 cartesian flatbottom xrest yrest zrest krest wrest
```

For a flatbottom restraint you must in addition specify the width of the flat region of the potential.

```
chunk restraint add id1-id2 bond harmonic krest
```

For a bond restraint you must specify only the force constant

```
chunk restraint add id1-id2-id3-id4 dihedral harmonic theta krest
```

For a dihedral harmonic restraint you must specify the target equilibrium angle and the force constant. This restraint does not work on solvent molecules and on protein backbone atoms.

The following example shows how to add a harmonic potential restraint between a ligand atom and a protein atom.

```
chunk id add 1 SOLUTE 1 N2 LI8
```

This chunk will create id number 1 which will point to solute atom 1 (the first atom in the solute pdb file), named c00, from residue L10.

```
chunk id add 2 PROTEIN 1 O 318
```

This chunk will create id number 2 which will point to protein pdb loaded as protein1 by ProtoMS. The atom named O in residue 318 will be selected. Note that 318 is the residue number that appear in the PDB file. It is not necessarily the 318th residue to be loaded by ProtoMS in this protein.

```
chunk restraint add 1-2 bond harmonic 5.0 3.33
```

This chunk will cause a restraint to be added between the atoms id 1 and 2 points to. The functional form of this restraint will be a harmonic potential that is function of the distance between these two atoms. The force constant will be 5 kcal mol - 1. A - 2 and the equilibrium distance 3.33 angstrom.

Applying a hardwall restraint is slightly different

```
chunk id add 1 SOLUTE 2 O00 WAT
```

This chunk will create id number 1 which will point to solute number 2, looking at the O00 atom of resname WAT

```
chunk hardwall 1 25.890 16.895 59.083 1.8 1000000000
```

This chunk will apply a hardwall restraint to the center of geometry of the solute number 2. The form of this restraint is spherical, with a radius of 1.8 and will be centered at the point defined by the coordinates 25.890 16.895 59.083. If the center of geometry of the molecule attempts to leave this radius then a huge penalty is applied, preventing the move. Equally, if any atom from another molecule tries to occupy the hardwall region then the penalty is applied.

A hardwall restraint can also be applied on the initial position of the center of geometry of a ligand. In this case, no coordinates need to be specified, and the lines results:

```
chunk hardwall 1 1.8 1000000000
```

This option should be quite useful when the ligand simply wants to be kept in its initial position.

5.4.5 Frequent output generation

Incidental generation of output files might not be convenient either for the production of results and restart files nor for PDB outputs. Consistently, there is an alternative option which allows for the generation of these files *while* the *simulate* chunk is running.

This is controlled with the alternative key word dump:

```
dump frequency command
```

This manner of output generation can be applied to all commands included in *Results and Restarts* secction, as well as *PDB output* section.

An example of a dump line would be:

```
dump 100000 results write results
```

This line, given as input for ProtoMS, will append results information to the *results* file every 100000 moves, thoughout the *simulate* part of your simulation.

It is important to note how the appending behaviour variates. For frequent results and PDB printing, new results will be appended to the existent file. However for the restart generation, the existing file will be overwritten every time and the old restart will be moved to another file. Consistently these imput lines:

```
dump 100 results write results
dump 100 pdb all file=all.pdb
dump 100 restart write restart
dump 100 averages reset
chunk simulate 400 solvent=10 solute=5 volume=1
```

Will generate four results reports all appended to the file *results*, four PDB conformations of the system appended to *all.pdb*, but only one restart report (the last printed) in the file *restart*.

Dump lines can be written in any order, and they all will be applied while the *simulate* chunk is running.

5.4.6 Miscellaneous

As well as running the simulation, there are also a collection of other things that you can do in a simulation chunk. These are

```
chunk singlepoint
```

Calculate the energy of the current system and output it to the SPENERGY stream. This is useful if you just want to use ProtoMS to evaluate a forcefield energy. You can set up the input files, turn off all streams, direct stream SPENERGY to STDOUT and run a simulation that only consists of this 'singlepoint' chunk.

```
chunk soluteenergy N
```

Calculate the energy of solute N. This calculates the energy of solute N and outputs the components of this energy in great detail. This is useful for debugging a forcefield or for collecting average energy components that are more finely divided than those normally collected.

```
chunk fakesim
```

Performs one step of simulation, without doing anything other than adding the energies to the averages. This can be useful for debugging purposes.

```
chunk retienergy 0.2
```

The RETI free energy method requires the calculation of the energy at the neighbouring two λ windows at the end of the simulation. This chunk will calculate the energy at λ windows 0.2 above and below the reference state, and will output the results to the RETI stream.

```
chunk lambda 0.5
```

Sets λ to 0.5. Will calculate and return the change in energy associated with this change in λ. This is useful if you wish to perform a slow growth or fast growth free energy simulation. You could also use this in conjunction with the 'averages print' and 'averages reset' chunks to calculate the free energy of all windows across λ within a single simulation. This is because the window widths are preserved by the change in λ, thus if the λ windows were 0.1 0.2 0.4 before the change, then they would be 0.4 0.5 0.7 after the change. Note that the values of λ are clamped between 0.0 and 1.0.

```
chunk lambda 0.5 0.6 0.4
```

As above, except set the λ values of the forwards and backwards windows to 0.6 and 0.4 respectively.

```
chunk lambda delta 0.1
```

As above except instead of directly setting λ, change λ by 0.1. This will also increase the value of λ for the for- wards and backwards windows by 0.1.

```
chunk freeenergy 0.3 0.5
```

Calculate quantities need for free energy estimators. This will calculate the derivative of the potential with respect to λ as needed for thermodynamic integration, and energies at λ =0.3 and λ =0.5 as needed for Bennett Acceptance Ratio method. All of these energies will be printed to the INFO and ENERGY streams.

```
chunk fixresidues 1 all
```

Fix all of the residues of protein 1.

```
chunk fixresidues 1 1-10 12 14 16-20
```

Fix the residues of protein 1. Only fix residues 1 to 10, 12, 14 and 16 to 20.

```
chunk fixresidues 1 none
```

Unfix all of the residues of protein 1.

```
chunk fixbackbone 1 all
```

Fix the backbone of all residues of protein 1. This chunk has the same syntax as the fixresidues chunk.

```
chunk fixbackbone 1 none 20-35
```

Unfix all of the residues of protein 1, then fix the backbone of residues 20-35. This ensures that only the backbone of residues 20-35 is fixed.

```
chunk12 transrot 1 0.0 0.0
```

Set the translation and rotation displacements for solute 1 to zero. This overrides the values read in the template file. The first floating point number is the translation displacement and the second one is the rotation displacement and is optional. Can be useful for pure solvent and gas-phase calculations.

```
setstream info=stdout move=off
```

Direct the INFO stream to STDOUT and turn the MOVE stream off.

```
setstream restart=myfile.txt warning=stderr
```

Direct the RESTART stream to myfile.txt and the WARNING stream to STDERR. solvate]

```
chunk solvent box xdim ydim zdim [xorig yorig zorig xmax ymax zmax]
```

This command can be used to replicate a solvent file loaded as solvent1 such that the final solvent occupies a box of dimensions xdim ydim zdim with origin (0,0,0). Alternatively the origin can be specified along with the maximum coordinates of the cubix box. solvate2]

```
chunk solvent cap xorig yorig zorig rad
```

As before but the output will be a spherical cap of solvent centered at the specified origin and with a radius rad. The last two commands can be used to create large solvent boxes when needed. Once this chunk has been performed, you should save a pdb of the system using the chunk pdb and then edit the output file such that it can load as ProtoMS solvent pdb. The process of replicating the solvent molecules can be quite memory consuming and you may find you have to recompile ProtoMS so that it can handle a large number of solvent molecules, particularly if the coordinates of the system you want to solvate are far away from the coordinates of the solvent molecules in the input solvent box.

5.5 Setup and analysis tools

As there are many options that can be set in ProtoMS, we provide a range of setup tools that can be used to setup the most common type of simulations. The main tool is called `protoms.py` and is document in the next chapter. For more advanced use, one can use the individual setup tools as documented in the tools chapter.

In order to perform analysis of ProtoMS simulations, there is a range of tools that can be used. They are documented in the tools chapter.

CHAPTER

SIX

INPUT FILES

ProtoMS can read in five types of input file

- **Parameter / Forcefield file** These provide the forcefield parameters used in a simulation, and the templates (z-matricies) that are used to specify the connectivity and flexibility of the simulated molecules.
- **Protein file** These are simple PDB format files that contain the coordinates of the protein chains to be simulated. Only one protein chain may be contained within each protein PDB file.
- **Solute file** These are simple PDB format files that contain the coordinates of the solutes to be simulated. Only one solute may be contained within each solute PDB file.
- **Solvent file** These are simple PDB format files that contain the coordinates of the solvent molecules to be simulated. Multiple solvent molecules may be contained within each solvent file.
- **Restart file** These are files used by ProtoMS to save and restore the coordinates of all of the molecules in the system.

ProtoMS is insensitive to case, so you can mix upper case and lower case within these files without affecting how they are read.

6.1 Parameter / Forcefield Files

The parameter file is the most powerful, and hence the most complicated of all of the input files read by ProtoMS. The parameter file provides all of the forcefield parameters that are used in a simulation, and it also provides all of the templates that provide the connectivity and z-matrices of all of the loaded molecules. The parameter file uses a word based format, meaning that you can leave as many spaces between words on a line as you like, and you do not have to worry about lining up data into particular columns.

The general format of a parameter file is shown below:

```
# comment lines start with a '#'

mode clj
#.... charge / Lennard Jones forcefield parameters

mode bond
#.... bond parameters

mode template
#.... templates

#parameter file uses a word-based format, so leave as many spaces as
#you want between words, e.g.
```

(continues on next page)

(continued from previous page)

```
       mode            clj

mode bond   #comments can also go at the end of any lines, like this!

MoDe DiHeDrAl  # you can use whatever case you want (though try to make
               # things readable!
```

How ProtoMS reads the parameter file is controlled by which `mode` the file has been set. There are several different modes, and as figure 3.5 shows, it is possible to change between modes within a single file. The different modes are

- **info** This mode is used to read in control information for the forcefield.
- **clj** This mode is used to read in the charge and Lennard Jones (clj) parameters for the simulation.
- **bond** This mode is used to read in the bond parameters for the simulation.
- **angle** This mode is used to read in the angle parameters.
- **ureybradley** This mode is used to read in the Urey-Bradley parameters.
- **dihedral** This mode is used to read in the dihedral parameters.
- **template** This mode is used to read in the templates (z-matricies) used in the simulation. The template format is quite complex, so is described in the next section.

ProtoMS will only read lines that are valid within the mode that is being read. If ProtoMS could not read a line, or finds an incorrectly formatted line, then ProtoMS will print a message to the WARNING stream and will skip that line. It is therefore very important that you check the WARNING stream if you are writing or modifying a parameter file. To help you, ProtoMS will write out detailed information about a loaded parameter file to the DETAIL stream. You should check this output to ensure that any changes you make to a parameter file are being correctly loaded by ProtoMS.

ProtoMS can be asked to load as many forcefield files as you desire. Each parameter or template within the forcefield files has either a numerical or name based ID. If two forcefield files have parameters or templates that share the same ID, then ProtoMS will use the value that was read last. ProtoMS will of course warn you that it has overwritten an earlier parameter (by outputting a message to the WARNING stream) but this behaviour could still trip you up! To help you, all of the parameters that use numerical IDs in the forcefield files supplied with ProtoMS use IDs that are between 1 and 2999. You can thus use numerical IDs that are greater than or equal to 3000 without worrying about a clash.

mode info

This mode is used to read in control information for the forcefield. This information is used to set parameters that affect which functions are used to evaluate the forcefield, and to set the values of forcefield-global parameters. The following lines are valid within this mode

```
ljcombine type
```

where *type* can be *arithmetic* or *geometric*. This sets the combining rules used for the Lennard Jones σ parameter to either the arithmetic mean (as used by AMBER), or the geometric mean (as used by OPLS). See equations (4.3) and (4.4) for the functional forms of these combining rules.

```
scl14coul float
```

This sets the 1-4 coulombic scaling factor, e.g. for OPLS the value should be 0.5 (see eq (4.9)).

```
scl14lj float
```

This sets the 1-4 Lennard Jones scaling factor, e.g. for OPLS the value should be 0.5 (see eq (4.9)).

mode clj

This mode is used to read in the charge and Lennard Jones (clj) parameters used by the simulation (see equations (4.1) and (4.11)). Only one type of line is valid within this mode

```
par id amber proton-number charge sigma epsilon
```

id is the unique identifying number for this clj parameter. This can be any number from 1 to MAXCLJ (by default this is 10000). If this ID is the same as an already read CLJ parameter, then ProtoMS will write a warning to the WARNING stream, and will overwrite the old CLJ parameter with the new parameter. To help prevent unintentional ID clashes, then the forcefields supplied with ProtoMS only use parameter IDs from 1 to 2000, and the solvent models supplied with ProtoMS use parameter IDs 2001 to 2999. You are thus free to use parameter IDs from 3000 in your own parameter files.

amber is the AMBER atom type associated with this clj parameter. The AMBER atom type is a two letter code that is used to identify the atom for the purposes of assigning bond, angle, dihedral or Urey-Bradley parameters. If this is a parameter for a dummy or non-chemical parameter, then the AMBER atom type should be '??'. Note that the AMBER type is case sensitive. This is different to other parts of ProtoMS, and is required as the GAFF forcefield uses case to distinguish between different AMBER types.

proton-number is the number of protons in the atom associated with this clj parameter, e.g. 1 for hydrogen, 6 for carbon or 8 for oxygen.

charge, *sigma* and *epsilon* are the partial charge (in |el), and Lennard Jones σ (A) and ϵ (kcal mol-1) parameters associated with this clj parameter, e.g.

```
par 2001 OW 8 -0.834 3.15061 0.1521 # TIP3P oxygen
```

specifies the clj parameter for oxygen in TIP3P water, with parameter number 2001, AMBER atom type 'OW' proton number 8, a partial charge of -0.834 |el, σ = 3.15061 A and ϵ = 0.1521 kcal mol-1 .

Parameter ID 0 is a special clj parameter used to represent a null atom. This null atom has charge, σ and ϵ values of 0.0, an AMBER atom type of 'DM' and a proton number of 0.

mode bond

This mode is used to read in the bond parameters used by the simulation. Two types of line are valid within this mode

```
par id force-constant bond-length
```

`id` is an identifying number from 1 to MAXBNDPARAM (default 5000) that is used to uniquely identify a bond. As in the case of the clj parameters, new parameters with the same ID number will overwrite old parameters with that ID number, and the parameter files supplied with ProtoMS will only use IDs from 1 to 2999, so you can safely use parameters 3000 and up.

`force-constant` is the force constant (k_{bond}, see eq (4.5)) for the bond parameter. The units of k_{bond} are kcal mol-1 A-2 . `bond-length` is the equilibrium bond length (r_0), in units of A.

The second type of line valid in this mode is used to associate a pair of AMBER atom types with a bond parameter

```
atm amb1 amb2 id
```

This line specifies the bond between atoms with AMBER atom types `amb1` and `amb2` is assigned the parameters from bond ID `id`. Note that this bond parameter does not need to have been loaded when this line of the parameter file is being read, as bond parameters are not assigned until after all parameter files have been read. If none of the bond parameter files provide this bond ID, then ProtoMS will print a message to the WARNING stream and will set the bond ID to 0. As in the case of the clj parameters, 0 is a special parameter used to specify a null bond, whose bond parameters, and thus energy, are all 0.0. In addition, any bond involving an AMBER atom with a null clj parameter (i.e. having AMBER atom type 'DM') will be automatically set to use bond parameter 0. It is not possible to have a non-null bond parameter for bonds that involve dummy atoms.

These bond atm lines are indexed by the AMBER pair amber1-amber2. If this AMBER pair has already been loaded then its parameter is overwritten with the new parameter. Note that bonds are symmetrical, thus bond index amb1-amb2 is equal to amb2-amb1.

mode angle

This mode is used to read in the angle parameters used in the simulation and its format and behaviour is almost identical to that used in the bond mode. Again, only two types of line are valid within the angle mode

```
par id force-constant angle-size
```

and

```
atm amb1 amb2 amb3 id
```

`id` is an indentifying number from 1 to MAXANGPARAM (default 5000) that is used to uniquely identify an angle parameters. `force-constant` is the force constant (k_{angle} , see eq (4.6)) for the angle parameter, in units of kcal mol-1 degree-2 . `angle-size` is the equilibrium angle size (θ_0) in units of degrees. Angle ID 0 is the null angle, and the forcefield files supplied with ProtoMS will only use angle IDs from 1 to 2999.

The atm line is again very similar to that in the bond mode, with in this case the angle between atoms with AMBER types `amb1-amb2-amb3` being assigned angle parameter id. Angles are also symmetric, so `amb1-amb2-amb3` is equivalent to `amb3-amb2-amb1`. Like the bond mode, any angle involving dummy atoms (AMBER type 'DM') will automatically be set to use the angle parameter 0. It is not possible to use a non-null angle parameter over an angle involving dummy atoms.

mode ureybradley

This mode is used to read in Urey-Bradley parameters (see eq (4.7)), and its format is identical to that of the angle mode. There are only two valid lines in this mode

```
par id force-constant uby-size
```

and

```
atm amb1 amb2 amb3 id
```

In this case force-constant refers to the Urey-Bradley force constant (k_{uby}), in units of kcal mol-1 A -2 and `uby-size` refers to the equilibrium Urey-Bradley length (x_0) in units of A. Everything else about this mode is identical to that of the bond mode.

mode dihedral

This mode is used to read in the dihedral parameters that are used in the simulation. There are three types of line that are value in this mode. The first of these is used to provide the parameters for a single dihedral cosine term :

term term-id k1 k2 k3 k4

`term-id` is an ID number from 1 to MAXDIHTERMS (default 5000) that uniquely identifies this dihedral cosine term. `k1` to `k4` are the values of the four constants (k_1 to k_4) that control the dihedral cosine term (see eq (4.8)). `k1` has units of kcal mol-1 , `k2` and `k3` are dimensionless, and `k4` is in units of degrees.

A full dihedral parameter is composed from the sum of individual dihedral cosine terms. The second valid line in the dihedral mode specifies which terms are associated with which parameters, e.g

```
par id 3 10 32
```

specifies that dihedral parameter `id` is formed as the sum of dihedral cosine terms 3, 10 and 32. You may specify as many dihedral cosine terms on this line as you wish from 1 to MAXDIHTERMSPERDIHEDRAL (default 6). As in

the bond, angle and ureybradley modes, `id` is a uniquely identifying number, in this case from 1 to MAXDIHPARAM (default 5000), with ID 0 referring to the special, null dihedral.

As in the case of the bond, angle and ureybradley modes, the AMBER atom set is used to associate dihedral parameters with actual dihedrals in a molecule. The final valid line associates the AMBER atom types of the four atoms in the dihedral with the dihedral parameter ID, e.g

```
atm amb1 amb2 amb3 amb4 id
```

Because dihedrals are symmetrical, `amb1-amb2-amb3-amb4` is equivalent to `amb4-amb3-amb2-amb1`.

mode born

This mode is used to read the Generalised Born parameters that are used in the simulation. A valid line is

```
par id atype iborn scalefac
```

where `atype` is an AMBER/GAFF atom type, `iborn` is an intrinsic born radius and scalefac a scaling factor for Pairwise Descreening Approximation calculations. These parameters have been optimised to be used with the AMBER or GAFF force fields.

mode surface

This mode is used to read surface area parameters that are used in the simulation. A valid line is

```
par id atype radius surftens
```

where `atype` is an AMBER/GAFF atom type, `radius` is the radius of the atom and `surftens` the surface tension of this atom type, which relates the solvent accessible surface area of this atom to a non polar energy. These parameters have been optimised to be used with the AMBER or GAFF force fields.

6.2 Templates

Templates are used to assign the z-matrix and forcefield parameters to loaded molecules. Templates are read in using the template mode of the parameter / forcefield file. Different types of template are used with the different types of molecules in ProtoMS

- **proteins** The backbone of each protein residue is assigned via a chain template. The sidechain of each residue is assigned via a residue template.
- **solutes** Solutes are assigned via solute templates.
- **solvents** Solvents are assigned via solvent templates.
- **GCsolutes** GCsolutes are assigned via GCsolute templates.

Chain Templates

Chain templates are used to assign the z-matrix and parameters of the backbone of protein residues. The start of a new chain template is indicated by the line

```
chain name
```

where name is the name of the chain template. This name uniquely identifies this chain template. If another chain template has been loaded with this name, then this chain template will overwrite it and a message will be output to the WARNING stream.

The valid lines that comprise a chain template are

```
bbatom id nam par0 par1
```

This line identifies which are the four bbatoms of the residue. `id` identifies which bbatom this atom is (from 1 to 4), `nam` gives the name of the atom (maximum of four characters), and `par0` and `par1` are the CLJ parameters for this atom at $\lambda = 0.0$ and $\lambda = 1.0$, and these must refer to a valid CLJ parameter (from 0 to MAXCLJ, default 10000). Note that CLJ parameter 0 is used to assign a dummy atom. The name of the atom is the same as that given in the PDB file for the protein, and is limited to a maximum of four characters. The atom name must uniquely identify the atom within the residue, so this name must not be used elsewhere within this chain template, or in any residue templates that connect to this chain template

```
atom nam par0 par1 bndnam angnam dihnam
```

This line identifies any extra atoms that are part of the backbone. `nam`, `par0` and `par1` have the same meanings as for the bbatom line. This is a z-matrix line, and `bndnam`, `angnam` and `dihnam` are the names of the atoms that are the reference from which the coordinates of this atom are generated (bond, angle and dihedral atoms). Note that this line does not state that there is a bond, angle or dihedral with these atoms. This line only says that these three atoms are used to construct this extra atom. Note that the atoms in a residue are built in sequence, so the bond, angle and dihedral atoms in this line must refer to atoms that were previously listed in the chain template.

```
zmat nam bndval angval dihval
```

This line provides the default values of the internal z-matrix coordinates for the atom called `nam`. `bndval`, `angval` and `dihval` are the default values of the bond length, angle size and dihedral size. This line is optional, and is only required if you either want ProtoMS to construct this atom if it is missing from the PDB file, or if you want ProtoMS to reset bond and angles to default values.

```
bond nam1 nam2
```

This line adds a bond between atoms name `nam1` and `nam2`. These two atom names must be present in the chain template. ProtoMS will not automatically add any bonds between atoms (except inter-residue bonds), so you must add all bonds that are present in the chain template. ProtoMS will use all of these explicitly added non-dummy bonds between atoms to generate all of the implicit angles and dihedrals within the backbone. Additional arguments may be present on this line to control the type of bond that is added, e.g.

```
bond nam1 nam2 dummy
```

adds a dummy bond between atoms `nam1` and `nam2`. A dummy bond is really a non-bond, as it has no energy, and its presence forces ProtoMS to treat atoms nam1 and nam2 as though they were not bonded together. You can make this bond flexible by adding the flex argument to the bond line, e.g.

```
bond nam1 nam2 flex delta
```

where `delta` is the maximum change in the bond length attempted in a Monte Carlo move in A. Note that you can only make degrees of freedom flexible if they are used in the construction of the z-matrix, i.e. atom `nam1` must be constructed via a bond with `nam2`, or `nam2` constructed from `nam1`.

The forcefield parameters for this bond will normally be assigned via the AMBER atom types of the constituent atoms. It is possible to override this assignment by explicitly assigning bond parameters, e.g.

```
bond nam1 nam2 param par0 par1
```

where `par0` and `par1` are the bond parameter IDs for this bond at $\lambda = 0.0$ and $\lambda = 1.0$. The bond parameter IDs must refer to valid bond parameters (0 to MAXBNDPARAM, default 5000), where parameter 0 is used to refer to a null bond. You can use parameter 0 to state that two atoms are bonded, but that the energy of the bond should not be evaluated, e.g.

```
bond nam1 nam2 param 0 0
```

Angles and Urey-Bradley terms and dihedrals in the chain template are specified almost identically as for the bond line, e.g.

```
angle nam1 nam2 nam3
```

adds an angle between atoms named `nam1-nam2-nam3`,

```
ureybradley nam1 nam2 nam3
```

adds a Urey-Bradley term between atoms named `nam1-nam2-nam3`, and

```
dihedral nam1 nam2 nam3 nam4
```

adds a dihedral between atoms named `nam1-nam2-nam3-nam4`. `dummy` and `param` options may be added to all of these lines, and `flex` may be added to the angle and dihedral lines (where delta is given in units of degrees).

ProtoMS uses the bonds specified in the template to work out where all of the implicit angles and dihedrals are in the backbone. You do not need to include implicit (additional) angles or dihedrals in the template file, and you can just the template to just the flexible angles and dihedrals. However there are some cases where you would not wish an implicit angle or dihedral to be evaluated, for example the dihedral energy may only need to be evaluated via one of the dihedrals around a bond, and not via any additional dihedrals. If this is the case then you will need to add those additional dihedrals to the template and use the dummy keyword to specify that these are dummy dihedrals and that their energy should not be evaluated.

It is not possible to add multiple bonds between the same pair of atoms, or multiple angles to the same triplet of atoms etc. ProtoMS will only use the first definition of a bond, angle, dihedral or Urey-Bradley term and will ignore any further attempts to set them.

As an example, the chain template for an amino acid backbone in the middle of a chain is as follows

```
#
#        --  HN       O  --
#         |   |       |  |
#   res-1| --N--CA--C--|res+1
#         |        |     |
#        --        X     --
#
mode template
chain aacenter
bbatom 1  N  3  3
bbatom 2  CA 6  6
bbatom 3  C  1  1
bbatom 4  O  2  2
atom  HN 4 4  N  CA  C
zmat  HN 1.010 119.8 180.0
bond  O  C
bond  C  CA
bond  CA N
bond  HN N
angle HN N CA flex 3.0
dihedral HN N CA C flex 3.0
# Now the parameters
mode clj
par 3 N  7 -0.570 3.250 0.170 # N, sp2 N in amide
par 6 CH 6  0.200 3.800 0.080 # CA, sp3 C with 1 H
```

(continues on next page)

(continued from previous page)

```
par 1 C   6   0.500 3.750 0.105 # C, carbonyl C
par 2 O   8  -0.500 2.960 0.210 # O, carbonyl O
par 4 H   1   0.370 0.000 0.000 # HN, amide hydrogen
```

This shows that it can be convienient to combine the chain template with the CLJ parameters for the template into a single parameter file.

Residue Templates

Residue templates are used to assign the z-matrix and forcefield parameters for the sidechains of protein residues. The format of a residue templates is almost identical to that of a chain template.

As an example, here is the residue template for OPLS united atom alanine

```
# ALANINE
#
#      N-CA-C
#          |
#         CB
#
mode template
residue ALA
info rotate 0.5 translate 1.0
backbone first aanterm middle aacenter last aacterm single aasingle
atom  CB  7 7 CA N C
zmat  CB  1.525 111.1 -120.0
bond  CB CA
angle CB CA N flex 0.5
#parameters
mode clj
par 7 C3 6   0.000 3.910 0.160 # CB, sp3 with 3 H
```

The start of a new residue template is signalled by the line

```
residue name
```

where name is the name of the residue template. This name uniquely identifies the template and because residues locate templates via the residue name, the residue template name is limited to a maximum of four characters. The lines that comprise a residue template are

```
info rotate rotdel translate trandel
```

This line provides information about the residue template. The option `rotate rotdel` specifies that the backbone rotation move would rotate the backbone by a maximum of `rotdel` degrees. The option translate `trandel` specifies that the backbone translation move would translate the backbone by a maximum of `trandel` A. Both of these options are optional, and may appear in any order on this line. If these options are not given, then the default translation and rotation values are both 0.0.

```
backbone position chain
```

This line states which chain templates are associated with this residue template for different positions of the residue within the protein, e.g.

```
backbone first aanterm middle aacenter last aacterm single aasingle
```

states that this residue template uses the chain template called `aanterm` if this was the first residue in the protein, `aacenter` if this residue was in the middle of the protein, `aacterm` if this was the last residue in the protein, and

`aasingle` if this was the only residue in the protein. You can place as many positions on this line as you wish, with possible positions being first, middle, last and single. You do not need to specify a chain template for every one of these positions, but ProtoMS will print a message to the WARNING stream if it needs a position that has not been specified.

The remaining lines in the residue template are the `atom`, `zmat`, `bond`, `angle`, `ureybradley` and `dihedral` lines, which have exactly the same meaning and formats as those in the chain template lines. Note that the names of atoms in the residue template must be different to those in any of its associated chain templates. Also note that you can (and indeed will have to!) refer to atoms that are present in the associated chain templates. In the example in above you can see that the only atom in the residue template is the united-atom 'CB', and that this is built from the 'CA', 'N' and 'C' atoms of its associated chain templates. This means that all of the chain templates associated with this residue template must include atoms named 'CA', 'N' and 'C'. If these atoms don't exist then ProtoMS will print many messages to the WARNING stream, and the simulation will fail.

ProtoMS will use the non-dummy bonds present in the residue template to find all of the implicit (additional) angles and dihedrals. If one of the bonds connect the sidechain to the backbone (one of the bonds should!), then the implicit angles and dihedrals between the sidechain and backbone will also be found. If you do not want the energy of these implicit angles and dihedrals to be evaluated then you need to specify them in the residue template with the `dummy` option set.

It is possible for a residue template to contain no atoms! While this may sound strange, it is necessary for residues such as glycine in united atom forcefields, or for some terminating residues (e.g. methylamine). The following example is the residue template for OPLS united atom glycine

```
# GLYCINE - this consists only of the glycine backbones
#
#    --C--CA--N--
#
mode template
residue GLY
info rotate 0.5 translate 1.0
backbone first glynterm middle glycenter last glycterm single glysingle
# glycine has no atoms, or internals!
```

Solute templates

Solute templates are used to assign the z-matrix and forcefield parameters to solute molecules. An example solute template for a united atom biphenyl is shown blelow

```
mode template   # make sure that the parameter file is being read in
                # in template mode

#                  |
#     CH3--CH2     |     CH2--CH3        Biphenyl, built as two residues,
#    /        \    |    /        \             PH1 and PH2
#  CH4        CH1-|-CH1         CH4
#    \        /    |    \        /        Note that each atom in a residue
#     CH5--CH6     |     CH6--CH5         must have a unique name but
#  PH1             |              PH2     that atoms in different residues
#                                         may have the same name
solute biphenyl
info translate 1.0 rotate 5.0

# Atoms in the first, PH1 residue
#
atom CH1 PH1  20 20  DM3 DUM  DM2 DUM  DM1 DUM  # First three atoms are built
atom CH2 PH1  20 20  CH1 PH1  DM3 DUM  DM2 DUM  # from the auto-generated
atom CH3 PH1  20 20  CH2 PH1  CH1 PH1  DM3 DUM  # dummy atoms (DM1-DM2-DM3)
```

(continues on next page)

(continued from previous page)

```
atom CH4 PH1  20 20  CH3 PH1  CH2 PH1  CH1 PH1
atom CH5 PH1  20 20  CH4 PH1  CH3 PH1  CH2 PH1
atom CH6 PH1  20 20  CH5 PH1  CH4 PH1  CH3 PH1

# Atoms in the second, PH2 residue
#
atom CH1 PH2  20 20  CH1 PH1  CH2 PH1  CH3 PH1
atom CH2 PH2  20 20  CH1 PH2  CH1 PH1  CH2 PH1
atom CH3 PH2  20 20  CH2 PH2  CH1 PH2  CH1 PH1
atom CH4 PH2  20 20  CH3 PH2  CH2 PH2  CH1 PH2
atom CH5 PH2  20 20  CH4 PH2  CH3 PH2  CH2 PH2
atom CH6 PH2  20 20  CH5 PH2  CH4 PH2  CH3 PH2
# Bonds between atoms - residue PH1
bond CH1 PH1  CH2 PH1
bond CH2 PH1  CH3 PH1
bond CH3 PH1  CH4 PH1
bond CH4 PH1  CH5 PH1
bond CH5 PH1  CH6 PH1
bond CH6 PH1  CH1 PH1
# interconnecting bond
bond CH1 PH1  CH1 PH2
# bonds in residue PH2
bond CH1 PH2  CH2 PH2
bond CH2 PH2  CH3 PH2
bond CH3 PH2  CH4 PH2
bond CH4 PH2  CH5 PH2
bond CH5 PH2  CH6 PH2
bond CH6 PH2  CH1 PH2

# only one flexible dihedral - interconnecting dihedral
dihedral CH2 PH2  CH1 PH2  CH1 PH1  CH2 PH1  flex 5.0
```

The format for a solute template is very similar to that of a residue template. The main difference is that while residue atoms are uniquely identified by thier atom name, solute atoms are uniquely identified by the combined atom name and residue name, e.g. the biphenyl atom CH2 PH2 is a different atom to CH2 PH1.

A new solute template is started with the line

```
solute name
```

where `name` is the uniquely identifying name of the solute template. As with the other templates, if a solute template with this name already exists, then it is overwritten by the new template. The name of the solute template can be any length up to 300 characters that can include spaces. Valid solute names thus include 'biphenyl' and 'test ligand 132B' Note that ProtoMS is insensitive to case, so it doesn't matter how you capitalise the solute name as ProtoMS will ignore it. The solute names 'biphenyl', 'BIPHENYL' and 'BiPhenyl' are all equivalent. ProtoMS will also strip the spaces before and after the solute name, and will replace multiple spaces within the name with single spaces, e.g. ' test ligand 132B ' is equivalent to 'test ligand 132B'.

The format and meaning of the valid lines in a solute template file are very similar to those of a residue and chain template. The line

```
info rotate rotdel translate trandel
```

has exactly the same format for a solute template as it does for a residue template, and the meaning is very similar. In this case this line sets the maximum amounts that the solute molecule as a whole will be rotated and translated by, in units of A and degrees respectively. This line is optional, and it is not present then the default maximum rotation and translation amounts are both zero. Note that translation and rotation of a solute is about the location of the first

automatically added dummy atom at the center of geometry of the solute.

```
atom nam res par0 par1 bnd bndres ang angres dih dihres
```

This line has a very similar meaning to the atom line of the residue and chain templates. In this case, this line identifies the solute atom called `nam`, in residue named `res`, and assigns it the CLJ parameters `par0` at $\lambda = 0.0$ and `par1` at $\lambda = 1.0$. The bond, angle and dihedral z-matrix atoms that are used to build this atom are the atom named `bnd` in residue `bndres`, the atom named `ang` in residue `angres` and the atom named `dih` in residue `dihres`. These z-matrix atoms must have appeared in the solute template before this atom. Note that this line does not add a bond, angle or dihedral between any of these atoms. The atom lines only specify how to move and construct the solute, not how to evaluate its energy.

```
bond nam1 res1 nam2 res2
```

This line adds a bond between solute atoms `nam1` in residue `res1` and `nam2` in residue `res2`. You can make this bond flexible by using the `flex` keyword in the same way as described for the chain and residue templates (as long as this bond is used in one of the atom z-matrix lines to construct one of the atoms). You can also use the same `dummy` keyword as the chain and residue templates to turn this into a dummy bond. As in those cases, a dummy bond is a non-bond, and has the effect of stating that the two atoms are not bonded together. The forcefield parameters for this bond are obtained via the AMBER types of the two solute atoms. However these parameters may be overridden through the use of the param keyword as used in the chain and residue templates, e.g.

```
bond nam1 res1 nam2 res2 param par0 par1
```

This line states that this bond uses bond parameter `par0` at $\lambda = 0.0$ and bond parameter `par1` at $\lambda = 1.0$. The angles, Urey Bradley terms and dihedrals in the solute are specified in a very similar manner

```
angle nam1 res1 nam2 res2 nam3 res3

ureybradley nam1 res1 nam2 res2 nam3 res3

dihedral nam1 res1 nam2 res2 nam3 res3 nam4 res4
```

the `dummy`, `flex` and `param` options may be used with these lines, with the exception of the `ureybradley` line, which cannot use the `flex` option. ProtoMS only uses the bonds listed in the solute template to work out which atoms are bonded together. ProtoMS does not try to guess which atoms are bonded together, so you will need to add all bonds that exist in the solute to the template file to ensure that the intramolecular energy is calculated correctly. ProtoMS will use these explicitly added, non-dummy bonds to work out all of the implicit (additional) angles and dihedrals in the solute. You do not need to include any additional angles or dihedrals in the solute template as they are added automatically by ProtoMS. If you do not want the energy of an additional angle or dihedral to be evaluated then you will need to add it to the template with the dummy option set. This is the same behaviour as in the chain and residue templates.

Solute templates have one extra type of valid line compared to chain or residue templates. This line is used to describe how the geometry of the solute changes with λ

```
variable nam res type val0 val1
```

`nam` and `res` are the name and residue of the atom that changes geometry with λ. `typ` can be either *bond*, *angle* or *dihedral* and describes whether the bond, angle or dihedral changes with λ, with `val0` giving its value at $\lambda = 0.0$ and `val1` giving its value at $\lambda = 1.0$. These variable geometry lines are very useful for free energy calculations where an atom is being 'switched off' by turning it into a dummy atom. You can use the variable geometry line to shrink the bond length to its z-matrix bonded atom, thus having the effect of pulling it within the van der waals sphere of the bonded atom. This prevents instabilities that may arise when the atom is close to being fully switched off.

Another use for variable geometry lines is to perform free energy calculations along structural coordinates, e.g. pulling two molecules apart. You can perform these sorts of calculations in ProtoMS by loading both molecules as a single

solute, with no bonds between the two molecules. You could then use a variable geometry line to change the distance between the two molecules with respect to λ.

Yet another use of geometry variation is to calculate the energy along an internal degree of freedom, e.g. by performing a torsion drive for the purposes of generating a dihedral forcefield parameter.

While λ may be used to change the forcefield parameters of any atom of any molecule in the entire system, only solutes may have their geometry changed with respect to λ. This is because geometry variations are implemented by making two copies of the solute and using these to shadow the original, reference solute. While you will not see these shadow solutes, they will reduce the number of solutes that you can load by two for every solute of variable geometry that you load. This means that while you can load a maximum of 50 solutes, you can only load a maximum of 16 solutes that have variable geometry.

Solvent Templates

Solvent molecules are implemented as rigid molecules in ProtoMS, so they do not require a z-matrix, nor do they have any internal degrees of freedom or energy terms. Solvent templates are thus much more simple than chain, residue and solute templates as they are only used to assign the forcefield parameters of the solvent molecules. An example solvent template for TIP4P water is shown in below

```
#
# TIP4P (T4P)
#
#     O00       dist(OH) = 0.9572 A
#   /  |  \     dist(OM) = 0.15 A
# H01 M03 H02   ang(HOH) = 104.52 deg
#

mode clj
par 2003  OW  8  0.000  3.15363  0.1550
par 2004  HW  1  0.520  0.0      0.0
par 2005  ??  0 -1.040  0.0      0.0

mode template
solvent T4P
info translate 0.15 rotate 15.0
atom O00 2003 2003
atom H01 2004 2004
atom H02 2005 2005
atom M03 2006 2006
```

A new solvent template is signified by the line

```
solvent name
```

where `name` is the uniquely identifying name of the solvent template. As in the cases of the other templates, if a solvent template with this name has been previously loaded, then it is overwritten. Solvent molecules are named using the residue name column from the PDB file, so the solvent name is limited to four characters. There are only two types of line that are valid within a solvent template. These are an info line, that has the same meaning as that in the solute templates, and

```
atom nam par0 par1
```

which states that the solvent atom called `nam` has CLJ parameters `par0` at λ = 0.0 and `par1` at λ = 1.0. The file *solvents.ff* in the *parameter* directory contains the solvent templates for a large number of standard solvents. All of the CLJ parameters used in this file range from 2001 to 2999.

GCsolute Templates

GCsolute molecules are implemented as rigid molecules in ProtoMS, like solvents, so they do not require a z-matrix, nor do they have any internal degrees of freedom or energy terms. GCsolutes templates are thus much more simple than chain, residue and solute templates as they are only used to assign the forcefield parameters of the GCsolute molecules. An example GCsolute template for TIP4Pg water is shown below

```
mode template
grand WAT
info translate 0.15 rotate 15
atom   O00  8003 8003
atom   H01  8004 8004
atom   H02  8004 8004
atom   M03  8005 8005
mode clj
#parameter atm proton-num charge(|e|) sigma(A) epsilon(kcal mol-1)
par  8003   OW    8    0.000      3.15363    0.1550
par  8004   HW    1    0.520      0.0        0.0
par  8005   ??    0   -1.040      0.0        0.0
```

A new GCsolute template is signified by the line

```
grand name
```

where `name` is the uniquely identifying name of the GCsolute template. As in the cases of the other templates, if a GCsolute template with this name has been previously loaded, then it is overwritten. GCsolute molecules are named using the residue name column from the PDB file, so the name is limited to four characters. There are only two types of line that are valid within a GCsolute template. These are an info line, that has the same meaning as that in the solute templates, and :

    atom nam par0 par1

which states that the GCsolute atom called `nam` has CLJ parameters `par0` at $\lambda = 0.0$ and `par1` at $\lambda = 1.0$. GCsolute templates should have values of CLJ parameters used in the files ranging from 8001 to 8999.

6.3 Automated Creation of Parameter and Template Files

Using an unsupported protein forcefield with ProtoMS requires the creation of a pair of parameter and template files. This process has been automated through a set of scripts that are able to import amber forcefield files and produce the necessary inputs for ProtoMS. For details of this please see the README in $PROTOMSHOME parameter/dev.

6.4 Protein File

Proteins are loaded from protein files. The names of the protein files are specified using the proteinN command described in section *Specifying input files*. The protein file is just a standard PDB format file. The name of the protein contained within this file is taken from the HEADER line of the PDB. e.g.:

```
HEADER p38 kinase
```

The protein name may contain spaces, though ProtoMS will strip any spaces before or after the name, and will collapse multiple spaces into a single space (much like it does with the solute name).

ProtoMS tries to follow the PDB format when it reads in PDB lines (see http://www.rcsb.org/pdb/docs/format/pdbguide2.2/guide2.2 frame.html). Atom names and coordinates are given on lines that start with ATOM or HETATM. As with the rest of ProtoMS, the capitalisation of these keywords is not important. Unlike the rest of ProtoMS, these lines have a strict format with respect to in which column each piece of data is recorded.

ProtoMS constructs the protein chain from the residue order that it reads in from the PDB file. This means that if a protein file contains residues numbered 5, 10 and 2, in that order, then ProtoMS will construct a protein chain with the sequence 5-10-2. ProtoMS will not try to be clever and numerically order your residues for you! One requirement when loading a protein PDB is that all atoms that are part of a residue are together within the PDB file. It is not possible to scatter atoms from one residue throughout the entire PDB file. In addition, all residues in the protein must have a unique residue number, and all atoms within the same residue must have unique names. ProtoMS loads the protein and assigns residue templates based on the residue names that it finds in the PDB filele. If ProtoMS cannot find a residue template that matches the residue name then it prints a message to the WARNING stream and then skips the residue. ProtoMS will use the residue and chain templates that it finds to work out which atom names should be present in the residue. If the PDB file provides an atom that matches the atom name, then ProtoMS assigns that atom from the template. If the PDB file does not provide an atom that matches the name, then if the atom name corresponds to one of the required bbatoms, then ProtoMS will print a severe message to the WARNING stream and will then skip the residue. If the missing atom is not a bbatom, then if the residue or chain templates provide zmat information for that atom then the coordinates for the atom are constructed automatically (and a message output to the WARNING stream). If no zmat information is available for this atom, then it is skipped and a severe message is output to the WARNING stream. Finally, if the PDB file provides an atom that is not part of the template, then that atom is skipped.

ProtoMS can only read a single protein chain from a PDB file. This means that you must split multi-chain PDB files into several files, and that PDBs using the 'A' or 'B' chain notation will be read incorrectly. If ProtoMS reads TER line, then it will print a message to the WARNING stream, and will then skip the rest of the PDB file. ProtoMS is capable of reading a wide variety of PDB files, and of fixing many of the errors that it encounters. Despite this, I would recommend that you do not just use a PDB direct from the databank, but that you first preprocess the PDB with another software package to ensure that the PDB is correct, and that polar hydrogens and titratable residues are included correctly.

6.5 Solute File

Solute input files are very similar to protein input files. Solute files are standard PDB format coordinate files. The name of the solute is read from the HEADER line in an identical manner to the name of a protein, e.g.:

```
header biphenyl
```

The solute name is used to locate the solute template, which is used to assign the z-matrix and forcefield parameters of the solute.

The solute PDB file has the same format as a standard PDB, with the requirements that all atoms belonging to a residue are together in the PDB, that each residue name is unique, and that all atom names within a residue are unique.

As is the case for protein files, ProtoMS will only read a single solute from each solute PDB file, and will skip the rest of the solute PDB if it encounters a TER line. It is intended that a future version of ProtoMS will remove this restriction.

ProtoMS will use the solute name to find the solute template for this molecule, and will then try to locate each atom from the template within the PDB file. If the atom does not exist then ProtoMS can automatically build the missing atom as long as its zmat information has been provided. If ProtoMS cannot build the atom then it skips it, after writing severe messages to the WARNING stream. If the PDB contains atoms that are not listed in the template then these atoms are ignored.

6.6 GCsolute File

GCsolute input files are very similar to protein input files, except that multiple GCsolutes can be loaded at once. GCsolute files are standard PDB format coordinate files. The name of each solvent molecule is taken from the residue name, and it is this name that is used to locate the template for each GCsolute molecule.

6.7 Solvent File

Solvent input files are very similar to protein and solute input files. Solvent files are standard PDB format coordinate files. Unlike the protein and solute files, many solvent molecules may be contained within each solvent input file. The name of each solvent molecule is taken from the residue name, and it is this name that is used to locate the template for each solvent molecule. ProtoMS will then try to locate each atom from the template within the PDB file. If the atom cannot be found then ProtoMS will write a severe message to the WARNING stream and will skip that atom. If the PDB contains atoms that are not part of the template then they are skipped. Note that ProtoMS will take the coordinates of the solvent molecule from the PDB file and will make no attempt to ensure that the internal geometry of the solvent molecule is correct for the template model (e.g. that TIP4P water has an O-H bond length of 0.9572 A).

If multiple solvent files are loaded, then the solvents from the newer files are appended onto the list of solvents loaded from the previous file. If solvent file 1 contains 340 solvent molecules, and solvent file 2 contains 10 solvent molecules, then the solvents from file 1 will be loaded as solvent molecules 1-340, and those from solvent file 2 will be loaded as solvent molecules 341-350.

Boundary conditions

As well as containing the coordinates of the solvent molecules, the solvent file may be used to specify the parameters needed for the boundary conditions. To do this, the solvent file must include a HEADER line that has one of the following formats

```
HEADER box dimx dimy dimz
```

This states that the solvent file contains a box of solvent of dimensions *dimx* A by *dimy* A by *dimz* A, with the box centered on the origin. Note that ProtoMS will not check to see if this information is correct, so you will need to ensure that that no solvent molecules lie outside of this box.

```
HEADER box ox oy oz tx ty tz
```

This states that the solvent file contains a box of solvent with the bottom-left-back corner located at coordinates (*ox*,'oy','oz') A and the top-right-front corner located at coordinates (*tx*,'ty','tz') A. Again ProtoMS will not check that this information is accurate!

```
HEADER cap ox oy oz rad k
```

This states that the solvent file contains solvent molecules restrained to be within a spherical cap of *radius* rad A, centered at coordinates (*ox*,'oy','oz') A, using a half-harmonic force constant of k kcal mol-1 A -2 . ProtoMS will not check to see whether or not this information is accurate.

Only one HEADER line may be included in each solvent file. How ProtoMS interprets these HEADER lines depends on which boundary conditions had been set for the simulation.

1. If no boundaries had been set for the simulation, then any information in the solvent files is ignored.
2. If a solvent cap had been set for the simulation, then any information in the solvent files is ignored and the solvent cap parameters are taken from the simulation parameter.
3. If a solvent box had been set then the solvent box dimensions are initially taken from the simulation parameter. However if any of the loaded solvent files specify the solvent box size then the solvent box dimensions are increased to encompass both the initial dimensions and the solvent box dimensions.

4. If 'solvent' boundaries had been set for the simulation then the boundaries used will be those obtained from the first solvent file that is loaded that contains a HEADER line. If none of the loaded solvent files contain a HEADER line then a warning is printed and no boundary conditions are used. Note that by default 'solvent' boundaries are set for all simulations. Warnings are printed if solvent files contain conflicting boundary types (e.g. specifying a box when a spherical solvent cap is used), or if multiple solvent files supply solvent cap parameters. If multiple solvent

files supply solvent box dimensions then the box is increased to the minimum size necessary to encompass all of the solvent boxes.

To make things simple, I recommend that you use one solvent file to describe your boundary conditions, and use the default option of specifying solvent boundaries via the solvent file (use `boundary solvent` in your command file, or do not supply a `boundary` value as `solvent` is the default).

ProtoMS will print out the boundary dimension to any output PDB file if that file contains solvent molecules.

6.8 Restart File

The restart file is used to save the coordinates of the entire system to a high precision such that they can be loaded up at a future point, or by another ProtoMS simulation. The format of the restart file is not yet fixed, so unfortunately there is the possibility that different versions of ProtoMS may not be able to read each other's restart files. This is considered a bug, and it is a development aim to stabilise the restart file format.

The restart file has deliberately been written as a human-readable text file. This means that the restart file is larger than it could be, but that it should be possible to manually edit a restart file, and understand its contents. If you wish to save space then I recommend that you compress the restart file via bzip2 or gzip. While the restart file is human-readable and editable, I recommend that you do not attempt to change the restart file unless you have a good understanding of the writerestart.F and readrestart.F source files that are used by ProtoMS to read and write them.

The restart file only contains the coordinates of the entire system and the parameters needed for the boundary conditions. This file does not contain energies or energy averages, as these are output via the RESULTS stream. The restart file does not contain information about the connectivity or setup of the system as these are contained in the command file and the protein, solute, solvent and parameter files.

You can write a restart file at any point during your simulation, and you can write as many restart files as you wish. This means that you can start your simulation with a bit of equilibration, and write a restart file for the final equilibrated configuration, and then run some production. This is a strategy used by many of the examples in the next chapter.

You can read a restart file at any point during your simulation, and you can read restart files as many times as you desire during a simulation. A restart file merely resets the coordinates of the system to those saved when the restart file was written. This means that you could run multiple chunks of a simulation from the same equilibrated configuration by reading in a restart file from the equilibrated configuration before performing each chunk of production. Note that you can only read a restart file into the same system that was used to write that restart file. If you try to load an incompatible restart file then the program will print lots of warnings and will probably close down!

CHAPTER

SEVEN

PROTOMS.PY

This program is used to setup a ProtoMS simulation. It was made with usability at highest priority. The only input that should be necessary is a couple of prepared PDB files containing the molecules one would like to simulate.

The program will create force field for small molecules, setup the protein and solvate the prepared system. At the moment it can setup the following types of simulations:

- Equilibration
- Sampling
- Dual-topology free energy
- Single-topology free energy
- Grand Canonical Monte Carlo (GCMC)
- Just Add Waters, stage 1 and 2 (JAWS-1, JAWS-2)

The program will create files and inputs based on experience that should work in most situations. However there might be situations where the created settings are not appropriate. One can then use individual tools to make a more custom setup, see this. One might also have to edit the files manually.

Syntax:

```
protoms.py [-s none|equilibration|sampling|dualtopology|singletopology|gcmc|jaws1|jaws
[-f folder1 folder2] [-p protein.pdb] [-sc scoop.pdb] [-l lig1.
pdb lig2.pdb ...] [-t template1 template2 ...] [-w water.pdb] [-c
cmdfile] [-r nrepeats | prefix] [--outfolder folder] [--atomnames
namefile] [--watmodel tip4p|tip3p] [--waterbox watbox] [--charge
charge1 charge2] [--singlemap mapfile] [--center cent] [--innercut
icut] [--outercut ocut] [--flexin sidechain|flexible|rigid]
[--flexout sidechain|flexible|rigid] [--scooplimit N] [--capradius
radius] [--lambdas nlambdas | lambda1 lambda2 ...] [--adams B1 B2 .
..] [--jawsbias bias] [--gcmcwater wat.pdb | N] [--gcmcbox box.pdb
| X Y Z A B C] [--nequil N] [--nprod N] [--dumpfreq N] [--absolute]
[--dovacuum] [--testrun] [--cleanup]
```

- **`-s none|equilibration|sampling|dualtopology|singletopology|gcmc|jaws1|jaws2` = the type of si**
  optional, default = `none`
- **`-f folder1 folder2` = name of folders to search for input files** optional, no default
- **`-p protein.pdb` = the name of the protein PDB file** optional, no default
- **`-o scoop.pdb` = the name of a protein scoop PDB file** optional, no default
- **`-l lig1.pdb lig2.pdb ...` = the name(s) of PDB file(s) containing ligand(s)** optional, no default

- **-t template1 template2 ... = the name(s) of ProtoMS template file(s) that needs to be loaded** optional, no default
- **-w water.pdb = the name of a PDB file with bulk water for the protein** optional, no default
- **-c cmdfile = the prefix for the created ProtoMS command file** optional, default = `run`
- **-r nrepeats | prefix = setup independent repeats of the simulation** optional, default = 1 `nrepeat` = repeats a created from 1 to `nrepeat` `prefix` = a single repeat is created, but prefix is appended to folders and files
- **--outfolder folder = the ProtoMS output folder** optional, default = `""` (empty string)
- **--atomnames namefile = the name of file containing conversion instructions** optional, no default if not given, takes the one in `$PROTOMSHOME/data`
- **--watermodel tip4p|tip3p = the water model to use** optional, default = `tip4p`
- **--waterbox watbox = the name a of a PDB file with a pre-equilibrated water box** optional, no default if not given, takes one in `$PROTOMSHOME/data`
- **--charge charge1 charge2... = the charges of the ligands** optional, default = 0
- **--singlemap mapfile = the correspondence map for single-topology setup** optional, no default
- **--center cent = the centre of the scoop** optional, default = 0.0,0.0,0.0
- **--innercut icut == the inner region cut-off in Angstroms** optional, default = 16.9 A
- **--outercut ocut == the outer region cut-off in Angstroms** optional, default = 20.0 A
- **--flexin sidechain|flexible|rigid = determine the flexibility of the inner region** optional, default = `flexible` `sidechain` = only the sidechains will be sampled in the simulation `flexible` = both sidechain and backbone will be sampled in the simulation `rigid` = no residues will be sampled
- **--flexout sidechain|flexible|rigid = determine the flexibility of the outer region** optional, default = `sidechain` `sidechain` = only the sidechains will be sampled in the simulation `flexible` = both sidechain and backbone will be sampled in the simulation `rigid` = no residues will be sampled
- **--scooplimit N = the minimum removed number of residues in a scoop** optional, default = 10
- **--capradius radius = the radius of the droplet solvating the protein** optional, default = 30
- **--lambdas nlambdas | lambda1 lambada2... = specification of λ; space for free energy calculations** optional, default = 16 if a single value is given, this number of λ-values is created uniformly from 0 to 1 if a list of values are given, this is the λ-values to use
- **--adams B1 B2... = the Adams parameter for GCMC** optional, default = 0
- **--jawsbias bias = the bias to apply in JAWS-2 simulations** optional, default = 0
- **--gcmcwater wat.pdb | N = the name of a PDB file with reservoir waters for GCMC and JAWS-1 or an integer** optional, no default if an integer is given this corresponds to the number of water to add to the GCMC/JAWS-1 box
- **--gcmcbox box.pdb | X Y X A B C = the name of a PDB file with GCMC or JAWS-1 simulation box dimension o** optional, no default if six numbers are given this corresponds to the origin (first three) and the length (last three) of the box
- **--nequil N = the number of equilibration moves** optional, default = 5E6
- **--nprod N = the number of production moves** optional, default = 40E6
- **--dumpfreq N = the frequency with which output is written to disc** optional, default = 1E5
- **--absolute = turns *on* the setup of absolute free energies** optional, default = off

- **`--dovacuum` = turns *on* the setup of vacuum simulation** optional, default = off
- **`--testrun` = turns *on* the setup of a short simulations appropiate for tests** optional, default = off
- **`--cleanup` = cleans up extraenous files and put them in a tar-ball** optional, default = off

Examples:

```
protoms.py
protoms.py.py -s sampling -l lig1.pdb --dovacuum --testrun
protoms.py -s dualtopology -l lig1.pdb lig2.pdb -p protein.pdb
protoms.py -s dualtopology -l lig1.pdb --absolute
protoms.py -s gcmc -p protein.pdb --adams -4 -2 0 2 4 6
```

Notes:

The program will try to locate previously created files for the protein and ligand in the current working directory or any folder specified with the `-f` flag. For ligands the program will replace `.pdb` with the appropriate ending, such as `.prepi` for Amber prepi files and `.tem` for ProtoMS template files.

Starting with just the PDB-files of the ligand(s) and the protein, the program will create the following files in the same folder as those PDB-files

- `lig.prepi` = the z-matrix and atom types of the ligand in Amber format
- `lig.frcmod` = additional parameters not in GAFF
- `lig.zmat` = the z-matrix of the ligand used to sample it in the MC simulation
- `lig.tem` = the complete template (force field) file for the ligand in ProtoMS format
- **`li1-li2.tem` = the combined template file of all ligands** the filename is a combination of the residue name of all ligands
- `lig_box.pdb` = the box of water solvating the ligand
- `protein_scoop.pdb` = the truncated protein structure
- **`protein_pms.pdb` = the original protein structure with ProtoMS naming convention** if the scoop removes to few residues, this file be created instead
- `water.pdb` = the cap of water solvating the protein system

In addition, for dual-topology simulations the following files are created: :

- **`lig1_dummy.pdb` = the dummy particle that the ligand will be perturbed to** only created if the –absolute flag is set

In addition, for single-topology simulations the following files are created:

- `li1-li2_ele.tem` = the ProtoMS template file for electrostatic single-topology perturbation
- `li1-li2_vdw.tem` = the ProtoMS template file for van der Waals single-topology perturbation
- `li1-li2_comb.tem` = the ProtoMS template file for combined/single-step single-topology perturbation
- **`settings.singlemap` = the created correspondance map for single topology** only named like this if the –singlemap argument is not set

In addition, for GCMC / JAWS-1 simulations the following files are created:

- `gcmc_box.pdb` / `jaws1_box.pdb` = the GCMC / JAWS-1 simulation box
- `gcmc_wat.pdb` = the GCMC / JAWS-1 reservoire waters
- `water_clr.pdb` = the cap of water solvating the protein system, cleared from the GCMC / JAWS-1 simulation box

In addition, for JAWS-2 simulations the following files are created:

- **`jaws2_watN.pdb` = the JAWS-2 water** each of the water given with the `--gcmc_water` flag will be written to an individual file
- `jaws2_notN.pdb` = the rest of the JAWS-2 water
- `water_clr.pdb` = the cap of water solvating the protein system, cleared from the GCMC / JAWS-1 simulation box

It will create at most three ProtoMS command files, one for the protein simulation, one for the ligand simulation and one for the gas-phase simulation. These can be used to run ProtoMS, e.g.

```
$PROTOMS/protoms3 run_free.cmd
```

Prerequisites:

The program assumes that both the ligand and the protein is prepared before. This includes for instance protonation. At the moment *only* Amber naming convention is supported.

The progam requires AmberTools to make force field for small molecules.

CHAPTER

EIGHT

TOOLS

In the `$PROTOMSHOME/tools` folder we have collect a range of useful scripts to setup and analyse ProtoMS simulations. Many of them are used by the `protoms.py` setup script. In this page we have collected the documentation for these tools with the user as a focus. Developers might be interested in looking at the Python code manual in the `.doc` folder.

8.1 ambertools.py

Examples:

```
ambertools.py -f benzene.pdb
ambertools.py -f benzene.pdb -n BNZ
ambertools.py -f benzenamide.pdb -c 0
ambertools.py -f benzene.pdb toluene.pdb -n BNZ TOL
```

Description:

This tool encapsulate the program `antechamber` and `parmchk` from the AmberTools suite of programs.

It will produce an Amber prepi-file, containing the z-matrix and atom types of the given solutes, parametrized with the general Amber force field and AM1-BCC charges. It will also produce an Amber frcmod-file with additional parameters not found in the GAFF definition. These files be named after the input `pdbfile`, replacing the extension `.pdb` with `.prepi` and `.frcmod`

The `antechamber` and `parmchk` program should exist in the system path or the AMBERHOME environment variable should be set correctly.

8.2 build_template.py

Examples:

```
build_template.py -p benzene.prepi
build_template.py -p benzene.prepi -f benzene.frcmod
build_template.py -p benzene.prepi -f benzene.frcmod -o benzene.template -n BNZ
build_template.py -p benzene.prepi -f benzene.frcmod -t 1.0 -r 10
```

Description:

This tool builds a ProtoMS template file for a solute given an Amber prepi file.

If the solute needs parameters not in the specified GAFF release, they should be supplied with the `frcmodfile`.

The tool will automatically make an appropriate z-matrix for Monte Carlo sampling. This works in most situations. However, if something is not working properly with the generated z-matrix, one can be supplied in the `zmatfile`

The default translational and rotational displacements are based on experience and should be appropriate in most situations.

8.3 calc_clusters.py

Examples:

```
calc_clusters.py -i all.pdb
calc_clusters.py -i all.pdb all2.pdb
calc_clusters.py -i all.pdb -o all_clusters.pdb
calc_clusters.py -i all.pdb -t complete
```

Description:

This tool cluster molecules from a simulation

It will extract the coordinates of all atoms with name equal to `atom` in residues with name equal to `molecule` in all input files and cluster them using the selected algorithm. If no atom is specified, the entire molecule will be clustered. By default this atom and residue name is set to match GCMC / JAWS output with the standard water template.

8.4 calc_density.py

Examples:

```
calc_density.py -i all.pdb
calc_density.py -i all.pdb all2.pdb
calc_density.py -i all.pdb -o gcmc_density.dx
calc_density.py -i all.pdb -r t4p -n o00
calc_density.py -i all.pdb -p 1.0 -s 1.0
calc_density.py -i all.pdb -e 0.5 -t gaussian
calc_density.py -i all.pdb -n 100
```

Description:

This tool discretises atoms on a grid, thereby representing a simulation output as a density.

It will extract the coordinates of all atoms with name equal to `atom` in residues with name equal to `residue` in all input files and discretise them on a grid. By default this atom and residue name is set to match GCMC / JAWS output with the standard water template.

The produced density can be visualized with most programs, e.g.

```
vmd -m all.pdb grid.dx
```

8.5 calc_dg.py

Examples:

```
calc_dg.py -d out_free/
calc_dg.py -d out_free1/ out_free2/ out_free3/ -l 0.1
calc_dg.py -d out_free1/ out_free2/ out_free3/ -u 0.9
calc_dg.py -d out_free1/ out_free2/ out_free3/ -e ti bar
calc_dg.py -d out_free1/ out_free2/ out_free3/ -e gcap
calc_dg.py -d out_free1/ out_free2/ out_free3/ --subdir b_-9.700 -e ti bar
```

Description:

This tool calculates free energies using the method of thermodynamic integration (TI), Bennet's Acceptance Ratio (BAR), multi state BAR (MBAR) and grand canonical alchemical perturbation (GCAP).

The program expects that in the `directory`, `directory2` etc. there exist an output folder for each λ-value, eg. `lam-0.000` and `lam-1.000`

The MBAR and GCAP estimators only works if PyMBAR is properly installed and can be loaded as a python library.

8.6 calc_dg_cycle.py

Examples:

```
calc_dg_cycle.py -d a_b/out_free -s + b_c/out_free -s + a_c/out_free -s - --
↪dualtopology
calc_dg_cycle.py -d a_b/out_free -s + b_c/out_free -s + a_c/out_free -s - --
↪singletopology comb
```

Description:

Calculates thermodynamic cycle closure for a set of simulations. This can be performed either for dual topology results, or single topology results. With single topology simulations, the electrostatic and van der Waals results can either be considered seperately `--singletopology sep` or together, `--singletopology comb`.

8.7 calc_gcap_surface.py

Examples:

```
calc_gcap_surface.py -d out_gcap -v 300.
calc_gcap_surface.py -d out_gcap --save-figures -v 300.
calc_gcap_surface --subdir b_-9.700 --estimators mbar -v 300.
```

Description:

8.7.1 Calculates the free energy from a surface-GCAP simulation. The volume of the GCMC region must be given using the `-v` flag. To calculate the free energy at a single B value, use the `--subdir` flag, and the energy can be calculated with any one dimensional free energy method.

8.7.2 calc_gci.py

Examples:

```
calc_gci.py -d out_gcmc/ -v 130.
calc_gci.py -d out_gcmc/ -v 130. -l 0.2
calc_gci.py -d out_gcmc/ -v 130. --save-figures
calc_gci.py -d out_gcmc/ -v 130. --pin_min
```

Description:

Collection of tools to analyse and visualise GCMC titration data of water using grand canonical integration (GCI). Used to plot average number of waters for a given Adams value, i.e. GCMC titration data, calculate transfer free energies from ideal gas, calculate absolute and relative binding free energies of water, calculate and/or estimate optimal number of bound waters. As described in Ross et al., J. Am. Chem. Soc., 2015, 137 (47), pp 14930-14943.

Error estimates of free energies and optimal number of waters are based on automatic repeated fitting of the ANN from different random initial parameters. This can be increased with `--nfits`.

8.8 calc_replicapath.py

Examples:

```
calc_replicapath.py -f out_free/lam-0.*/results -p 0.000 1.000
calc_replicapath.py -f out_free/lam-0.*/results -p 0.000 0.500 1.000 -o replica_paths.
↪png
calc_replicapath.py -f out_free/t-*/lam-0.000/results -p 25.0 35.0 45.0 -k temperature
```

Description:

This tools plots the path of different replicas in a replica exchange simulation as a function of simulation time.

If the kind of replicas is from λ replica exchange the `replica1` and `replica2` etc should be individual λ-values to plot.

If the kind of replicas is from REST or temperature replica exchange the `replica1` and `replica2` etc should be individual temperatures to plot.

8.9 calc_rmsd.py

Examples:

```
calc_rmsd.py -i benzene.pdb -f out_bnd/all.pdb -r bnz
calc_rmsd.py -i benzene.pdb -f out_bnd/all.pdb -r bnz -a c4
```

Description:

This tool calculate the RMSD of a ligand in a simulation.

If the `atom` name is given, the tool will calculate the RMSD of that atom with respect to its position in `pdbfile`. Otherwise, the program will calculate the RMSD of the geometric centre with respect to `pdbfile`.

A force constant to keep the ligand restrained for free energy calculations is estimated from the RMSD using the equipartition theorem.

8.10 calc_series.py

The tool will estimate the number of independent samples for a given observable in the production part using the method of statistical inefficiency. The equilibration time will also be estimated from a method that maximizes the number uncorrelated samples as suggested on alchemistry.org.

Apart from the raw series, the tool can also plot the running average if the `--average` flag is set or the moving average if the `--moving` flag is used.

Typically only a single ProtoMS results file will be analysed and plotted. However, for the series `grad` and `agrad` (the gradient and analytical gradient, respectively), multiple results file can be given. In this case, the gradients for each results file is used to estimate the free energy using thermodynamic integration.

8.11 calc_ti_decomposed.py

Examples:

```
calc_ti.py -d out_free/
calc_ti.py -d out_free/ -l 0.1 -u 0.9
calc_ti.py -b out_bnd/ -d out_free --dualtopology
calc_ti.py -d out_free -g out_gas
```

Description:

This tool calculates free energies of individual energetic components using the method of thermodynamic integration (TI).

The program expects that in the `directory` there exist an output folder for each λ-value, eg. `lam-0.000` and `lam-1.000`

Block estimates can be constructed by combining `-l` and `-u`. For instance, these commands calculates the free energy while incrementally increasing the equilibration

```
for X in `seq 0.0 0.1 1.0`
do
calc_ti_decomposed.py -d out_free -l $x
done
```

8.12 clear_gcmcbox.py

Examples:

```
clear_gcmcbox.py -b gcmc_box.pdb -s water.pdb
clear_gcmcbox.py -b gcmc_box.pdb -s water.pdb -o water_cleared.pdb
```

Description:

This tool clears a GCMC or JAWS-1 simulation box from any bulk water placed there by the solvation method.

In a GCMC and JAWS-1 simulation the bulk water is prevented to enter or exit a GCMC or JAWS-1 simulation box. Therefore, bulk water that are within this box needs to be removed prior to the GCMC or JAWS-1 simulation.

The `boxfile` is typically created by `make_gcmcbox.py` and the `waterfile` is typically created by `solvate.py` and can be either a droplet or a box.

8.13 convertatomnames.py

Examples:

```
convertatomnames.py -p protein.pdb
convertatomnames.py -p protein.pdb -c $PROTOMSHOME/data/atomnamesmap.dat
convertatomnames.py -p protein.pdb -s charmm
```

Description:

This tool converts residue and atom names to ProtoMS convention.

This script modfies in particular names of hydrogen atoms, but also some residue names, e.g. histidines.

A file containing conversion instructions for amber and charmm is available in the `$PROTOMSHOME/data` folder.

8.14 convertwater.py

Examples:

```
convertwater.py -p protein.pdb
convertwater.py -p protein.pdb -m tip3p
convertwater.py -p protein.pdb --ignoreh
```

Description:

This tool converts water molecules to a specific model.

Currently the script recognizes TIP3P and TIP4P water models. The valid values for `style` is therefore `t4p, tip4p, tp4, t3p, tip3p, tp3`

If the `--ignoreh` flag is given, the script will discard the hydrogen atoms found in `pdbfile` and add them at a random orientation.

8.15 distribute_waters.py

Examples:

```
distribute_waters.py -b 53.4 56.28 13.23 10 10 10 -m 12
distribute_waters.py -b 53.4 56.28 13.23 10 10 10 -m 12 --model t3p --resname T3P
distribute_waters.py -b 53.4 56.28 13.23 10 10 10 -m myonewater.pdb --number 12 -o␣
↪mywatersinbox.pdb
```

Description:

This tool can place water molecules at random within a GCMC or JAWS-1 simulation box.

It can place molecules in random positions and orientations with their geometry center restricted to the given dimensions of a box.

8.16 divide_pdb.py

Examples:

:: divide_pdb.py divide_pdb.py -i mypmsout.pdb -o individual -p outfolder/

Description:

This tool splits up a PDB file with multiple models (the keyword END defines the end of a model) into several PDB files.

8.17 generate_input.py

Examples:

```
generate_input.py -s dualtopology -l lig1.pdb lig2.pdb -p protein.pdb -t li1-li2.tem -
↪pw droplet.pdb -lw lig1_wat.pdb --lambas 8
generate_input.py -s dualtopology -l lig1.pdb dummy.pdb -t li1-dummy.tem -lw lig1_wat.
↪pdb --absolute
generate_input.py -s gcmc -p protein.pdb -pw droplet.pdb --adams -4 -2 0 2 4 6 --
↪gcmcwater gcmc_water.pdb --gcmcbox gcmc_box.pdb
generate_input.py -s sampling -l lig1.pdb -t lig1.tem --dovacuum
```

Description:

This tool generates input files with commands for ProtoMS.

The settings generate are made according to experience and should work in most situations.

The tool will create at most two ProtoMS command files, one for the protein simulation and one for the ligand simulation. These can be used to run ProtoMS, e.g.

```
$PROTOMS/protoms3 run_free.cmd
```

8.18 make_dummy.py

Examples:

```
make_dummy.py -f benzene.pdb
make_dummy.py -f benzene.pdb -o benzene_dummy.pdb
```

Description:

This tool makes a matching dummy particle for a solute.

The dummy particle will be placed at the centre of the solute.

8.19 make_gcmcbox.py

Examples:

```
make_gcmcbox.py -s benzene.pdb
make_gcmcbox.py -s benzene.pdb -p 0.0
make_gcmcbox.py -s benzene.pdb -o benzene_gcmc_box.pdb
```

Description:

This tool makes a GCMC or JAWS-1 simulation box to fit on top of a solute.

The box will be created so that it has the extreme dimensions of the solute and then `padding` will be added in each dimension

The box can be visualised with most common programs, e.g.

```
vmd -m benzene.pdb benzene_gcmc_box.pdb
```

this is a good way to see that the box is of appropriate dimensions.

When an appropriate box has been made, it can be used by `solvate.py` to fill it with water.

8.20 make_single.py

Examples:

```
make_single.py -t0 benzene.tem -t1 toluene.tem -p0 benzene.pdb -p1 toluene.pdb
make_single.py -t0 benzene.tem -t1 toluene.tem -p0 benzene.pdb -p1 toluene.pdb -m␣
↪bnz2tol.dat
make_single.py -t0 benzene.tem -t1 toluene.tem -p0 benzene.pdb -p1 toluene.pdb -o bnz-
↪tol
```

Description:

This tool makes ProtoMS template files for single topology free energy simulations.

The program will automatically try to match atoms in `template0` with atoms in `template1`. It will do this by looking for atoms with the same atom type that are on top of each other in `pdbfile0` and `pdbfile1`. A cut-off of 0.02 A2 will be used for this. All atoms that cannot be identified in this way are written to the screen and the user has to enter the corresponding atoms. If no corresponding atom exists, i.e., the atom should be perturbed to a dummy, the user may enter blank.

The user may also write the corresponding atoms to a file and provide it as `map` above. In this file there should be one atom pair on each line, separated by white-space. A dummy atom should be denoted as `DUM`. If `map` is not given, the program will write the created correspondence map to a file based on the `outfile` string.

Currently, dummy atoms are not supported in the solute at $\lambda = 0.0$. Therefore, this solute needs to be the larger one.

The tool will write two ProtoMS template files, one for the electrostatic perturbation, one for the van der Waals perturbation and one for the combined perturbation. These template files will end in `_ele.tem`, `_vdw.tem`, `_comb.tem` respectively.

A summary of the charges and van der Waals parameters in the four states will be printed to the screen. This information should be checked carefully.

8.21 merge_templates.py

Examples:

```
merge_templates.py -f benzene.tem dummy.tem -o bnz-dummy.tem
```

Description:

This tool combines several ProtoMS template files into a single template file.

The force field parameters in `file2` will be re-numbered so that they do not conflict with `file1`. This is important when you want to load both parameters into ProtoMS at the same time.

8.22 plot_theta.py

Examples:

```
plot_theta.py -m WA1 --skip 50
plot_theta.py -m WA1 -p theta_wa1
```

Description:

This tool plots the theta distribution resulting from a JAWS stage one simulation.

Two different histograms will be generated. One in which all different copies of the same molecule are added up, and a different one where each copy is displayed individually.

8.23 scoop.py

Examples:

```
scoop.py -p protein.pdb
scoop.py -p protein.pdb  -l benzene.pdb
scoop.py -p protein.pdb  --center "0.0 0.0 0.0"
scoop.py -p protein.pdb  --center origin.dat
scoop.py -p protein.pdb  --innercut 10 --outercut 16
scoop.py -p protein.pdb  --exclude 189 190
scoop.py -p protein.pdb  --added 57 58 59
```

Description:

This tool truncates a protein and thereby creating a scoop.

All residues outside `ocut` is removed completely. `icut` is used to separate the scoop model into two different regions, that possibly can have different sampling regimes. The sampling regimes are determined by `--flexin` and `--flexout`.

If the user would like to finetune the residues in the scoop this can be done with `--excluded` to discard specific residues or `--added` to include specific residues.

The scoop will be centred on the `ligandfile` is such a file is provided. Otherwise, it will be centred on the flag `--center`. The argument to this flag can be either a string with three numbers specifying the centre, as in example three above. It can also be the name of a file containing the centre, as in example four above.

Crystallographic waters that are in `proteinfile` will also be truncated at `ocut`

The PDB file will contain specific instructions for ProtoMS to automatically enforce the values of `--flexin` and `--flexout`.

8.24 solvate.py

Examples:

```
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -s benzene.pdb
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -s benzene.pdb -p 12.0
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -s benzene.pdb -pr protein.pdb -g␣
↪droplet
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -s benzene.pdb -pr protein.pdb -g␣
↪droplet -r 24.0
```

(continues on next page)

(continued from previous page)

```
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -pr protein.pdb -g droplet -c 0.0
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -pr protein.pdb -g droplet -c "0.0 10.
↪0 20.0"
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -pr protein.pdb -g droplet -c "76 86"
solvate.py -b $PROTOMSHOME/data/wbox_tip4p.pdb -s gcmc_box.pdb -g flood
```

Description:

This tool solvates a ligand in either a droplet or a box of water. It can also flood a GCMC or JAWS-1 simulatios box with waters.

Pre-equilibrated boxes to use can be found in the `$PROTOMSHOME/data` folder.

To solvate small molecule it is sufficient to give the `solutefile` as in the first example above. This produces a box with at least 10 A between the solute and the edge of the water box, which should be sufficient in most situation. Use `padding` to increase or decreas the box size as in the second example. The solvation box is created by replicating the pre-equilibrated box in all dimensions and then removing waters that overlap with solute atoms.

To solvate a protein in a droplet, specify `proteinfile` and `droplet` as in the third example above. This produces a droplet with radius of 30 A, which was choosen to work well with the default options in `scoop.py`. Use `radius` to obtain a smaller or larger droplet as in the fourth example. The centre of the droplet can be on a ligand if `ligandfile` is specified. Otherwise, the `center``argument is used. This argument can be either ``cent` (the default) that places the droplet at the centre of the protein. It can also take a single number as in the fifth example above in case it is placed at this coordinate in all dimensions. It can also take a string with three numbers which is the origin of the droplet in x, y, and z dimensions, see the sixth example above. If two numbers are given as in the seventh example above, it is assumed that this is an atom range and the droplet will be placed at the centre of these atoms. The droplet is created by putting random waters from the pre-equilibrated box on a grid, displacing them slightly in a random fashion.

The tool can also be used to fill a box with waters for GCMC and JAWS-1 simulations, similar to `distribute_waters.py`. In this case the solute is typically a box created by `make_gcmcbox.py` and `flood` needs to be specified, see the last example above. This gives a box filled with the bulk number of waters.

8.25 split_jawswater.py

Examples:

```
split_jawswater.py -w waters.pdb
split_jawswater.py -w waters.pdb -o jaws2_
```

Description:

This tool splits a PDB file containing multiple water molecules into PDB files appropriate for JAWS-2.

For each water molecule in `pdbfile` the tool will write a PDB file with individual water molecules named `outprefix+watN.pdb` where N is the serial number of the water molecule. Furthermore, the tool will write a PDB file with all the other molecules and name if `outprefix+notN.pdb` where again N is the serial number of the water molecule. In these latter PDB-files, the water residue name is changed to that of the bulk water, e.g., `t3p` or `t4p`.

For instance, if `waters.pdb` in the second example above contains 3 water molecule, this tool will create the following files:

```
jaws2_wat1.pdb
jaws2_wat2.pdb
```

(continues on next page)

(continued from previous page)

```
jaws2_wat3.pdb

jaws2_not1.pdb
jaws2_not2.pdb
jaws2_not3.pdb
```

CHAPTER

NINE

TEST SUITE

The ProtoMS test suite can be found in the `$PROTOMSHOME/tests` directory. It contains a set of Python scripts and all required input files and reference output files to run a sanity check on the ProtoMS code, both the source (Fortran) code and the (Python) tools. In this page you will find a list of the different tests, a brief indication of which part of ProtoMS each of the tests is checking and instructions to run each of the individual tests separately, or all of them as a whole.

9.1 Dependencies

The Python module `nose` is required to run the test suite. You can find more information on nose on its website `nose.readthedocs.org/en/latest/`.

9.2 Running all tests

The simplest and recommended way to run the tests is to run the command `ctest` while in the build directory `$PROTOMSHOME/build`. This will run all tests and report the success or failure of each. For more information use the command `ctest -V` which will print all output from the tests as well as output from both the Python and Fortran components of ProtoMS.

If ProtoMS was compiled without MPI, the following test scripts will not be run:

- `test_mpi_install.py`
- `test_gcmc.py`
- `test_jaws2_sim.py`
- `test_reti_sngl.py`
- `test_reti_dbl.py`

9.3 Individual tests

In this section you will a list of all tests, with a brief explanaition of which part of the ProtoMS code they are testing.

Most of the test scripts define multiple individual tests, a setup test and a simulation test. From the ProtoMS build directory (i.e. `$PROTOMSHOME/build`), to run all tests within a single test script use `python ../tests/test_<scriptname>.py`. To run only a single test stage use `python ../tests/test_<scriptname>.py <FullTestName>` or `ctest -R <FullTestName>`

9.3.1 List of tests

A list of all Python scripts that correspond to each of the tests is shown below:

- `test_install_dependencies.py`
- `test_ligand_setup.py`
- `test_parameters_ff.py`
- `test_path.py`
- `test_tools_protoms.py`
- `test_prot_setup.py`
- `test_equil.py`
- `test_energies.py`
- `test_sampling.py`
- `test_jaws1.py`
- `test_mpi_install.py`
- `test_gcmc.py`
- `test_jaws2.py`
- `test_reti_sngl.py`
- `test_reti_dbl.py`

9.3.2 test_install_dependencies.py

Coverage

This test covers the requirements for the installation of ProtoMS. It checks that the AmberTools are installed and available at `$AMBERHOME` and that the Python modules numpy, scipy and matplotlib are available.

9.3.3 test_ligand_setup.py

Coverage

Checks that the set up of ligands with the ProtoMS tools generates the expected results.

Reference Data Location

`$PROTOMSHOME/tests/setup`

9.3.4 test_parameters_ff.py

Coverage

Checks that all expected parameter files are found in `$PROTOMSHOME/parameter`.

9.3.5 test_path.py

Coverage

Checks that `$PROTOMSHOME` has been set correctly.

9.3.6 test_tools_protoms.py

Coverage

Checks that all expected Python scripts corresponding to the ProtoMS tools are present in `$PROTOMSHOME/tools`.

9.3.7 test_prot_setup.py

Contains * ProtSetupTest

Coverage

Checks that the set up of protein and ligand with the ProtoMS tools generates the expected results.

Reference Data Location

`$PROTOMSHOME/tests/setup/`

9.3.8 test_equil_prot.py

Contains * EquilSetupTest * EquilSimulationTest

Coverage

Checks both setup and run of the `equilibration` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/equil/`

9.3.9 test_energies.py

Contains * EnergiesSimulationTip3pTest * EnergiesSimulationTip4pTest

Coverage

Checks the generation of correct energies for different water models used as solvent.

Reference Data Location

`$PROTOMSHOME/tests/energies/`

9.3.10 test_sampling.py

Contains * SamplingSetupTest * SamplingSimulationTest

Coverage

Checks both setup and run of the `sampling` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/sampling/`

9.3.11 test_jaws1.py

Contains * Jaws1SetupTest * Jaws1SimulationTest

Coverage

Checks both setup and run of the `jaws1` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/jaws1/`

9.3.12 test_mpi_install.py

Coverage

Checks that MPI is available for running simulations requiring it.

9.3.13 test_gcmc.py

Contains GcmcSetupBoxTest GcmcSetupTest GcmcSimulationTest

Coverage

Checks both setup and run of the `gcmc` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/gcmc/`

9.3.14 test_jaws2.py

Contains * Jaws2SetupTest * Jaws2SimulationTest

Coverage

Checks both setup and run of the `jaws2` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/jaws2/`

9.3.15 test_reti_sngl.py

Contains * RetiSnglSetupTest * RetiSnglSimulationFreeTest * RetiSnglSimulationGasTest

Coverage

Checks both setup and run of the `singletopology` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/RETI_sngl/`

9.3.16 test_reti_dbl.py

Contains * RetiDblSetupTest * RetiDblSimulationTest

Coverage

Checks both setup and run of the `dualtopology` simulation type among those offered by `protoms.py`.

Reference Data Location

`$PROTOMSHOME/tests/RETI_dbl/`

INDEX

R

S

T

U

V