User Guide to Osprey Dissipative Particle Dynamics Simulation Code

Author: Julian C Shillcock

Version History: 1.0 June 2019
Updated: 1.01 August 2020
Updated: 1.1 October 2020

Contents

---

1) Introduction

This document describes the OSPREY-DPD Dissipative Particle Dynamics simulation code. It should be read through in sequence to learn how to use the code. A short introduction to the theory of DPD is given at the end, but a longer description, and detailed applications, can be found in these references:

R. D. Groot and P. B. Warren, *J Chem. Phys.* **107**:4423-4435 (1997)

P. Espagnol and P. B. Warren, *J. Chem. Phys.* **146**:150901 (2017)

J. C. Shillcock and R. Lipowsky, *J. Chem. Phys.* **117**:5048-5061 (2002)

J. C. Shillcock and R. Lipowsky, *Nature Materials* **4**:225 - 228 (2005)

J. C. Shillcock, *Langmuir* **28**:541-547 (2012)

J. C. Shillcock, M. Brochut, E. Chénais, J. H. Ipsen, *Soft Matter* **16**:6413 (2020)

2) Code and file conventions

The executable code will be referred to as *dpd* although you may rename it.

The single input file is called the *Control Data File* or just the *input file*. Its name must be of the form:

dmpci.nnn

where nnn is referred to as the *runID* and should be replaced by a user-defined alphanumeric string. Throughout this document, any field of the form "nnn" must be replaced by a user-specified alphanumeric string that contains a combination of letters and numbers, the underscore _ and hyphen - characters. No other characters are allowed - especially not spaces. Sometimes this string must start with a letter and this will be made clear at that point.

All output files produced by the code will have the string nnn embedded in their names. Files that are produced more than once during a simulation will also have the simulation time at which they were produced in their names.

Commands that should be entered into a terminal window are shown as follows (text after the // sequence is a comment in this document and should not be typed)

> ls   // lists the contents of the current directory on linux

> cd myDir   // change to the directory containing the dpd code

> ./dpd 123   // execute the dpd code using the input file "dmpci.123"

3) Installing the code

Download the source code from github, compile and link it for your platform, and place it in a directory that is in your path or in the directory where you will be executing it.

git@github.com:Osprey-DPD/osprey-dpd.git

See the README.md in the github repo for details.

4) Running the code at the command line

The dpd executable is executed at the command line. How to do this differs for each platform.

Linux

Open a Terminal Window, navigate to the directory containing the executable, and enter the run command at the prompt as described below under 1) Single run".

Mac OS X

Open a terminal window (double-click the Terminal application in "Macintosh HD/Applications/ Utilities), navigate to the directory containing the executable and enter commands described below under 1) Single run".
.

Windows

Open a Command Window, navigate to the directory containing the executable and enter commands described below under 1) Single run". See this web page for discussion of command windows:

https://www.lifewire.com/how-to-open-command-prompt-2618089

1) Single run

On all platforms, the input file must have a name of the form "dmpci.nnn" where "nnn" is a user-defined alphanumeric string - called the runId - used to name the output files.

NB. Only letters, numerals (0-9) and "-" and "_" characters are allowed in the runId. Definitely no spaces.

The code is executed by entering its name and providing the runId of an input that is in the current directory as an argument to the command.

Example

 if the input file"dmpci.123" is in the current directory, the simulation is started by executing the following command at the prompt (Note that the prompt is represented here by the > character).

> ./dpd 123

NB ONLY the extension of the input file must be specified in the run command, i.e., the user-defined string after the "." character.

All output files are produced in the *same directory as the input file* and will have the user-specified extension embedded in their names to identify them.

To place the simulation in the background on linux/Mac, so you can continue to use the terminal window, place an ampersand character "&" after the command before pressing "Enter":

> ./dpd 123 &

If the code is executed without arguments it will prompt for an input file as follows; enter the extension and hit return:

```
> ./dpd

DPD Experiment Code 1.5
Enter a runId (nnn):
> 123
```

An estimate of how long a particular simulation will take in real time can be made by multiplying the number of beads in the simulation (bead density * box volume) by the number of time steps and dividing by $2*10^{10}$. The result is the number of cpu-days required.

Two other ways of executing the code follow.

2) Generating a default input file

If the runId "nnn" is passed to the code but NO corresponding input file is found in the current directory, the code will create a default input file with the name dmpci."nnn". The user can then edit this file as desired. This is useful in case you have no input file at hand. By default, this file has a simulation that is only 20 time steps long so it should finish quickly.

3) Multiple runs in series

Multiple runs can be executed in series in one command by passing the list of runIds on the command line separated by spaces:

```
> ./dpd  100 101 102
```

This requires that input files dmpci.100, dmpci.101, dmpci.102 be present in the current directory.

---

5) Input file

The input file uses Keyword/Value pairs for the data items needed.

All keywords and values in the input file **are case sensitive** and must appear in the order shown.

No comments or extra text can appear anywhere in this file except in the section for Commands, where a special command is available to insert comments into the sequence of commands. See Section 8 for more details.

When a string in the input file is enclosed in inverted commas, e.g., " Water/surfactant bilayer ", there **MUST** be a space before and after each inverted comma character. The parser won't recognise the start or end of the string without these spaces.

Here is a complete input file for a lipid membrane self-assembly simulation followed by an explanation of the various parameters needed. The contents of the input file are between the starred lines. Note that the file MUST end in one or more newline characters.

Comments in red are NOT part of the file.

```
// ************************************************************************

dpd       // Selects the simulation type - don't modify

Title    " Water/surfactant bilayer "
Date     26/04/17                 // Date format must be as shown
Comment " H3(T4)2 lipids.
         Unstretched bond length of 0.5, k3 = 15
         A/N = 1.255,  1631  lipids
         Box 32**3 - Repeat of 1002 for consistency check "

State    random                // Name of the initial state type

// List of bead types, each line holds the following data:
// Name of bead must begin with a letter
// Radius of all beads is 0.5 - don't modify
// Conservative force parameter
// Dissipative force parameter

Bead  H
      0.5
      25
      4.5

Bead  T
      0.5
      50    25
      4.5   4.5

Bead  W
      0.5
      35    75      25
      4.5   4.5     4.5


// List of bond types: Beads 1, 2 the Hookean spring constant
// and the unstretched length of the spring

Bond  H H  128   0.5
Bond  H T  128   0.5
Bond  T T  128   0.5

// List of stiff bond types: Beads 1, 2, 3, the bending constant
// and preferred angle

BondPair  H T T  20.0  0.0
BondPair  T T T  20.0  0.0

// List of polymer types: Name, number fraction, and molecular
// shape of the polymer

Polymer Water  0.9801083   " (W) "
```

```
Polymer Lipid  0.0198917   " (H H (* (T T T T)) H T T T T) "


Box      32 32 32  1 1 1  // Box size and CNT cell size
Density     3            // Average bead density
Temp        1            // Temperature
RNGSeed     -712094      // Seed for random number generator
Lambda      0.5          // Constant, see Groot/Warren paper
Step        0.005        // Integration step size
Time        1000         // Total simulation time
SamplePeriod    100      // No of steps between samples
AnalysisPeriod  500      // No of steps between averaging
DensityPeriod   1000     // No of steps between density fields
DisplayPeriod   100      // No of steps between snapshots
RestartPeriod   1000     // No of steps between restart states
Grid    10 10 256        // Size of analysis grid

Analysis                 // Turn analysis of bilayer on
    Type    bilayer
    Times   0 1000
    Polymer Lipid
    Normal  0 0 1
    Slice   256
    Grid    16 16
    Solvent Water


// List of commands to execute during the run

Command ToggleBeadDisplay           1  W
Command SetCurrentStateCamera        1  0.5 -0.5 -0.5  0.5 0.5 0.5
Command SetCurrentStateDefaultFormat 1  Paraview

Command SetTimeStepSize              100  0.01

// ***************************************************************
```

Simulation type and Header Information

The first string in the input file must be "dpd" in lower case (no blank lines before it).

The **Title** and **Comment** strings are for convenience, you can enter anything between the inverted commas. The **Date** is useful for searching input files for runs executed at a certain time.

Initial State specification

See Section 6 for a description of allowed initial states.

Bead type specification

The set of bead types that are present in the simulation is specified next.

A bead type must specify its name, radius (fixed at 0.5), and the conservative and dissipative force parameters for ALL previously-defined bead types and itself. Bead names must begin with a letter, contain only letters or numbers, and must be unique.

NB While bead and polymer names are case sensitive, so that "W" and "w" represent different bead types, it is NOT a good idea to name them like this.

Each new bead type added will have one more pair of interaction parameters than the previous one. The last value on each line is the self-interaction. So the set of values looks like a lower-diagonal matrix.

The **conservative force parameter** can have any non-negative value. But specifying very large values (> 100) may lead to instabilities unless a very small integration step-size is used. Typical values are between 5 and 100.

The **dissipative force parameter** can be left at 4.5 for all bead types. See Groot and Warren (1997) if you want to know the consequences of changing this parameter.

NB Bead types specified do not have to be used in the simulation. You can create extra bead types for later use. But if a bond or polymer specifies a particular bead type then it must be present.

Bond type specification

The set of bonds that tie beads together into polymers is specified next.

A bond type is defined by specifying the two bead types it connects. Bonds are symmetric, so it is not necessary to specify two bonds to connect beads of different types.

All pairs of beads matching these types that occur in any polymer will be connected with a bond of the given type. The name of a bond, which is needed if you want to change the bond parameters with a command, is just the bead names concatenated, e.g., a bond connecting beads H to T will be called HT.

Bonds are Hookean springs with the force law

$F(x) = -k_2(x - x_0)$

where k_2 is the spring constant (128 in the example above), and x_0 is the unstretched length of the spring (0.5 in the example above above), and x is the separation of the bonded beads.

The spring constant can be any non-negative value, and is typically around 100 to ensure bonded beads do not separate too far. The unstretched length can be left at 0.5 unless very soft springs are desired in which case larger values can be used.

NB Bond types specified here do not have to be used in the simulation. But if a stiff bond specifies a particular bond type then it must be present.

Stiff Bond type specification

Stiff bonds are defined as two adjacent bonds in a polymer that have an energy associated with bending the bonds away from their preferred angle. A stiff bond is defined by specifying the names

of the three bead types that form it. The name of a stiff bond type is the names of these beads concatenated, e.g., a chain of T beads that occurs in the lipid tails above would have a stiff bond name TTT.

The energy associated with a stiff bond is:

$$V(\theta) = k_3 (1 - \cos(\theta - \theta_0))$$

where k_3 is the bending energy, θ_0 is the preferred angle (zero means the bonds are parallel in the minimum energy state), and θ is the angle between the two bonds.

NB Stiff bond types that are specified do not have to be used in the simulation but their component bonds and beads must exist.

Polymer type specification

The polymer (or molecule) types that make up the system to be simulated are specified next.

Each polymer is defined on one line, and requires its name, its number fraction, and the shape or architecture of the polymer. Polymer names must begin with a letter. The number fraction of a polymer is the ratio of the number of polymers of that type to the total number of polymers of all types. The number fractions of all polymer types must add up to 1.

The *shape string* of a polymer is a linear representation of the connectivity of the beads in the polymer. A polymer's shape must begin and end with actual bead names, and these are taken as the *Head* and *Tail* of the polymer and are used to define various properties such as the end-to-end length. All bead names and multiplier/branching/looping characters must must be separated from each other by spaces, and the whole shape string must be enclosed in inverted commas (with adjacent spaces). Bead names may be adjacent to brackets but it is a good idea to use spaces to make the string easily readable.

It is possible to redefine the Head and Tail beads for a polymer so that they are not the first and last beads. In this case, the beads specified must be unique in the polymer's shape string. Here is an example:
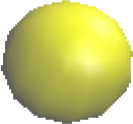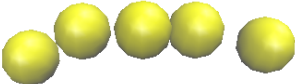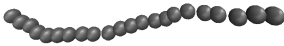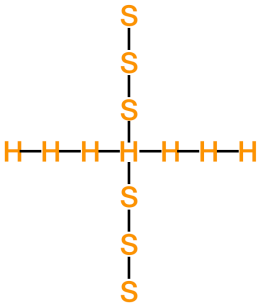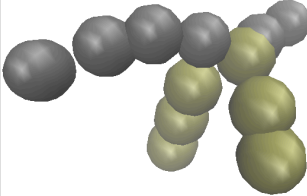
Polymer Lipid 0.01 " (H H1 (* (T T T T)) H (* (T T T T1)) H) " Head H1 Tail T1
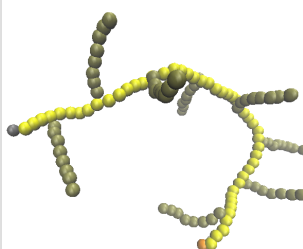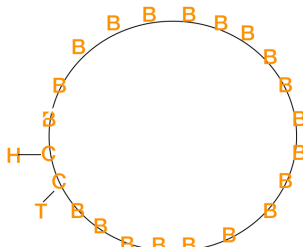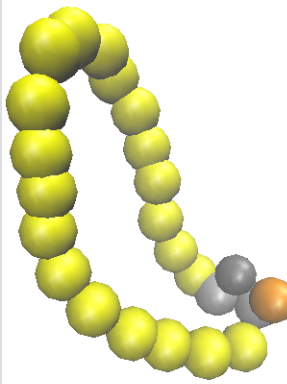
This represents a lipid molecule with 4 head beads and two tails. The H1 bead is the Head of the molecule and the T1 bead is the Tail. The end-to-end length of the molecule will be defined by the distance between the H1 and T1 beads at any given time.

As well as linear molecules, one can define side-chains and loops. These are specified using the special characters "*" and "/" respectively as shown in the examples below. A branch defines a sub-sequence of the polymer that branches off the previous bead in the shape string. Once the branch has been created in the code, the succeeding sequence of beads will continue to grow from the bead just prior to the branch. A loop allows a polymer to be connected to itself. The first bead that occurs in with the "/" character is the anchor and the second bead associated with the "/" character and the same number is connected to it.

Branches, loops and linear sequences of beads may be recursive, so a side-branch may contain a loop that itself contains branches that contain further branches, etc.

Here are examples of polymer shapes. Note that a polymer's shape string cannot start or end with the numeric multiplier, branch (*), or loop (/) character so an extra bead is often used to define the first and last beads in the polymer.

Shape String	Structure	Image	Description
" (W) "	W		*Polymer* containing a single bead W that represents, for example, water.
" (O O O O O) "	O-O-O-O-O		Linear molecule made up of 5 beads of type O.
" (H (20 H) H) "	H-H-H-H-H-H-H- -H-H-H-H-H-H-H- -H-H-H-H-H-H-H		Linear polymer made up of 22 "H" beads, with a numeric multiplier to avoid having to write out H many times. Note that the first and last beads must be specified explicitly, i.e., ((22 H)) is NOT allowed.
" (H H H (* (S S S)) (* (S S S)) H H H) "	S | S | S | S | H-H-H-H—H-H-H | S | S | S		Cross-shaped molecule with two side-chains "SSS" connected to the third "H" bead. The "*" character tells the parser that the next bracket is a side-chain.

" (H (8 (B B B (* (S (6 S) S)) B B B)) T) "	Too hard to draw!		*Comb* polymer with a backbone (48 B beads) and eight side-chains (8 S beads) regularly spaced along it. Note the H, T beads at the ends .
" (H (/ I C) (20 B) (/ I C) T) "			Ring-shaped molecule in which the two beads called C are connected to each other. The first "/" character defines an anchor point and the second one connects to it. More rings can be defined by using (/ 2 D) pairs, where D is a different bead type, and so on. Note the H and T beads that are required at the start and end of the shape string.

Simulation box and time specification

The size of the simulation box in the x, y, z dimensions is specified by the **Box** keyword. The first three values are the number of CNT cells in each dimension and the second three values are the widths of these cells in units of the bead diameter. You don't need to be concerned with the definition of the CNT cells, just leave the second three values at unity.

Typical box sizes are "10 10 10" for a small simulation or "32 32 32" for a medium sized one. Boxes of "40 40 40" or larger will take a long time to simulate.

The bead **Density** keyword specifies the average bead density in the simulation box. The total number of beads created in the simulation box is nearly equal to the density times the volume of the box. It is approximate because if some of the polymers have many beads, the code may not be able to create the exact number required by the number fractions, and it will round down the number of polymers to ensure an integer number are made.

The **Temp** keyword should be left at unity. The temperature may be changed, but this is advanced functionality that requires careful analysis to use.

RNGSeed is the seed for the random number generator. It must a negative integer, and should be different for each simulation. If this value is not changed, and the same input file is run several times, **exactly the same results** will be obtained on the same platform. Hence, to collect statistically significant results, this value must be changed between simulations.

Lambda is a parameter defined by Groot and Warren in their Velocity-Verlet integration scheme and described in their paper of 1997. It should be left at 0.5.

Step is the integration step size for the simulation. Smaller values are more accurate, but take more time. If too large a value is used, the simulation will be unstable. **Typical values are in the range 0.001 - 0.04**. See Groot and Warren for a discussion of the effects of step size on a simulation.

If the initial state contains bonds or stiff bonds with large force constants, it is a good idea to start with a small value of **Step**, e.g., 0.001, and use a command to change it to a larger value, e.g., 0.02 after the system has evolved for a few thousand time steps. See the Commands section for details.

Time is the total number of time steps in the simulation. It should be a round number to make the values of the sampling periods sensible. The actual simulation time used depends on the system being simulated, in particular how fast it approaches equilibrium and what statistical accuracy is desired.

There are constraints on the various sampling periods to ensure that the time-averaged analysis takes place regularly at the same frequency, and that all files are written at integer time steps.

SamplePeriod specifies the number of time steps between taking samples of observables that are to be time-averaged. It must be an integer divisor of the **Time** value and the **AnalysisPeriod** value.

AnalysisPeriod specifies the number of steps between writing out time-averaged observables. It must be an integer divisor of the **Time** value. For good statistical accuracy there should be at least 100 samples per analysis period.

DensityPeriod should be set equal to **Time**. This value can be ignored.

DisplayPeriod specifies the number of time steps between saving current state snapshots, and must be an integer divisor of **Time**. If it is set too small, a large number of files may be produced (**Time/ DisplayPeriod** to be precise).

RestartPeriod specifies the number of time steps between saving restart states, and must be an integer divisor of **Time**. As for DisplayPeriod, if it is set small, a large number (**Time/RestartPeriod**) of files will be produced. Unless you need extra restart states (e.g., if you think the simulation might exceed its allowed time before completing), you can safely set **RestartPeriod** equal to **Time**. This will produce a single restart state at the end of the simulation.

Grid is a parameter that controls the size of a 3D rectangular grid that is used in some analysis options. Unless needed as described in Section 10, it should be set to "1 1 1".

Analysis

See Section 10 Analysing the results of a simulation.

Commands

See Section 8 Issuing commands to a simulation.

6) Initial State types

Random

```
State    random
```

The most common initial state is a random distribution of all polymers throughout the simulation box. The code ensures that beads that are bonded together in a polymer are positioned close to their associated bonds' unstretched length to prevent large forces occurring in the initial state.

Restart

```
State    restart
RunId    100
StateId 10000
```

A restart state is a special form of initial state that continues a simulation from a previously saved state. It is the only initial state that requires more than one input file. In addition to the new input file, which we call "dmpci.101" here, the original input file (dmpci.100) and the specified restart state file (saved at time 10000) must be present in the current directory. The original runId is specified as the value of the keyword "RunId" and the time at which the restart state was saved is the value of the keyword "StateId". This time is embedded in the name of the restart files.

For this example, the following files must be present in the run directory:

```
New Input file:      dmpci.101
Old Input file:      dmpci.100
Old restart file:  dmpcrs.100.con.10000.dat
```

Some of the parameters in the new input file (e.g., simulation time, sample periods, RNG seed, analysis options, and commands) can be changed,. But many of them cannot: the box size, density, number and types of beads, bonds, stiff bonds, and polymers cannot be changed. This is a consequence of the simulation being carried out in the NVT ensemble: the numbers of particles, volume and temperature are constant.

NB. If the conservative/dissipative interactions between beads, or bond or stiff bond parameters are to be changed, this **must** be done by issuing commands in the restarted run to do so. The values present in the new input file are ignored.

Single-component bilayer membrane
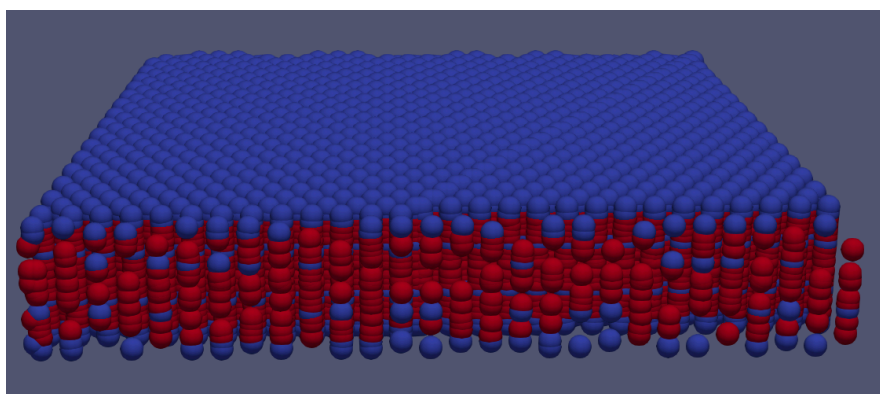

```
State    lamella
         Polymer         Lipid
         Normal          0 0 1
         Centre          0.5
         Thickness       5.0
         Linearise       1
         UpperFraction   0.5
         Polymerise      0
```

This places all the polymers specified by the **Polymer** parameter in a planar bilayer arrangement with all the other polymers randomly distributed throughout the remainder of the simulation box. The bilayer contains two monolayers, has its normal in the direction specified by **Normal** (it can only be in the x, y, or z directions), its centre at the point along the normal axis specified by **Centre** (as a fraction of the simulation box size in that dimension). The initial thickness of the membrane is specified by **Thickness**, (in units of the bead diameter) but as the simulation evolves the actual thickness will relax to its equilibrium value that may be different. An estimate of this value can be made by multiplying the number of beads along the polymer by the unstretched bond length used and doubling this.

The **Linearise** parameter is a boolean flag (0/1) showing if the beads in the polymers should be initially placed in a linear sequence or slightly randomly. As seen in the above snapshot, the initial placement of the polymers is very regular as each molecule is placed at the vertices of a triangular lattice in the plane of the bilayer. The polymers will fluctuate at the start of the simulation as the system evolves towards its equilibrium state.

UpperFraction specifies what fraction of the polymers used in the membrane should be placed in the upper monolayer. If this is not 0.5 the bilayer will be asymmetric, with more polymers in one monolayer than the other. The **Polymerise** keyword should be ignored and left at 0.

Multi-component bilayer membrane
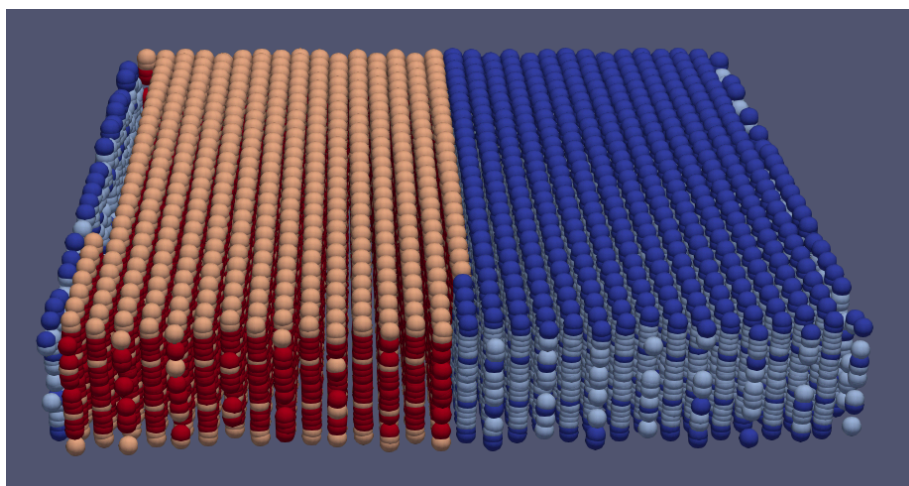

```
State    compositelamella
         Polymers     Lipid CoLipid
         Normal          0 0 1
         Centre          0.5
         Thickness       5.0
         Linearise       1
         UpperFraction   0.5  0.5
         Patches         1  1
         Polymerise      0
```

This is similar to the "lamella" initial state but any number of polymer species can be placed in the membrane. The geometric parameters are the same as above, but there are two differences:

UpperFraction - this parameter must be specified for **each polymer type** in the membrane. It allows different polymers to be distributed between the two monolayers independently of each other.

Patches - this new keyword is a boolean flag (0/1) with a value for **each monolayer.** If set to 0, all polymers in that monolayer will be randomly arranged throughout the monolayer, while if set to 1, all polymers of a given type will be positioned in the monolayers according to their order in the Polymers keyword.

Single-component vesicle
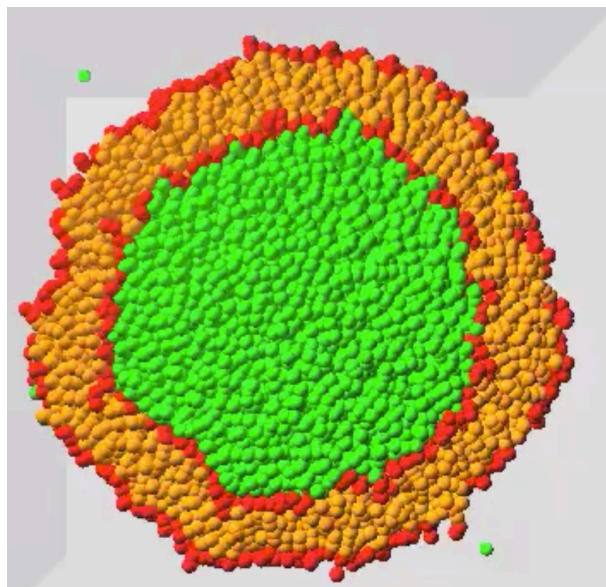

```
State    vesicle
         Polymers       Lipid
         Interior       Water1
         Centre         0.5 0.5 0.5
         OuterRadius    10
         Thickness      4.0
         OuterFraction  0.7353
         Patches        0  0
         Polymerise     0
```

Here, the polymers specified by the **Polymer** parameter are arranged as a spherical vesicle. In the image above, only half of the vesicle is shown. There may be more than one polymer type specified for the vesicle. Because the interior of a vesicle is topologically distinct from the exterior, the polymer types that are to be placed inside the vesicle must be specified separately as the values of the **Interior** keyword. Again, there may be several polymer types specified here. All other polymer types defined in the input file will be randomly distributed in the surrounding space. NB The same polymer type must NOT be specified in more than one region.

 The vesicle contains two monolayers, and its centre is at the point specified by **Centre** (as a fraction of the simulation box size). The outer radius of the vesicle is specified by **OuterRadius**, and the membrane thickness in **Thickness** (both in units of the bead diameter). As the simulation evolves the thickness will relax to its equilibrium value. The **OuterFraction** parameter specified what fraction of each polymer type in the vesicle will be placed in the outer monolayer. Because of the finite thickness of the bilayer, the number of molecules in the outer monolayer is larger than that in the inner monolayer, so this parameter is generally specified greater than 0.5. One can create an asymmetric vesicle by changing this parameter. For example, if a minor component is to be all placed in the outer monolayer, one would set the corresponding value to 1. There must be as many values assigned to this parameter as there are polymer types in the vesicle. The Patches keyword specifies if the polymers should be randomly distributed in the inner and outer monolayers of the vesicle (Patches 0 0), or occur in discrete patches in the order they polymers are specified (Patches 1 1). Note that the two values refer to the inner and outer monolayers respectively. The **Polymerise** keyword should be ignored and left at 0.

There are many other initial state types. Contact the author for more details.

---

7) Output files

A typical simulation produces a number of different types of output file. Depending on how often the data are analysed, or snapshots of the simulation state or restart states are saved, there can be a large number of files produced. The file names all start with "dmpc" and have 2 or more characters appended to this to indicate the type of data they contain.

The following types of output file are always produced (assuming the input file was dmpci.nnn, and ttt is the integer simulation time at which the corresponding file was saved):

dmpcas.nnn   - **Analysis State** contains a set of time-averaged observables

dmpccs.nnn.con.ttt,vtk   - **Current State** snapshots of the simulation (only bead x, y, z and
type)

dmpchs.nnn   - **History State** contains time series of various observables

dmpcis.nnn   - **Initial State** contains a copy of all the input parameters

dmpcls.nnn   - **Log State** contains information, warnings and error messages

dmpcrs.nnn.con.ttt.dat   - **Restart State** contains all the data needed to restart a run

There are several types of Current State format, but the one used in the course is suitable for visualising using the free Paraview software. Go here to download it - https://www.paraview.org.

Other recognised formats are **Povray** (www.povray.org), and a special format called **SolventFree** that includes connectivity information of all polymers so that offline analysis can be performed (use the command SetCurrentStateDefaultFormat to select the format.)

---

8) Issuing commands to a simulation

Commands are used to modify the execution of a simulation and turn on and off various analysis and output options.

Commands must be placed at the end of the input file (after any analysis options) and they all have the common specification:

Command <commandName>   <executionTime>   arg1 arg2 …

where <commandName> is the (case sensitive!) name of the command; <executionTime> is the time-step at which the command is to be executed; and any remaining fields are arguments required by the command.

Not all commands have arguments, while some commands will require more than one line to specify all their arguments. But every command must start on a new line.

NB Any number of commands may share the same execution time, but commands must be ordered as a series of non-decreasing execution times.

NBB Commands are executed at the **start** of the time-step specified as their execution time. Analysis options and saving snapshots, etc are performed at the **end** of their respective time-steps. So if a command is executed in the same time step as writing a snapshot it will take effect before the data is written.

Example: the following command toggles on/off the appearance of the "W" bead type in the current state snapshots at simulation time 1. It is useful when a simulation is mainly water so that the other polymers can be seen in the snapshots. This command may be issued any number of times: each appearance toggles the state from its previous value.

Command ToggleBeadDisplay   1   W

If appropriate arguments are not supplied, or are misspelled, or have illegal values, an error message is printed to the screen or log state file (depending on when the command executes).

List of Commands

There are three categories of command that can be issued during a simulation. They are described in the following three tables in alphabetical order.

The following conventions apply to all commands and their arguments:

1) All commands must specify their time of execution as the **first argument**. The tables below do not include the execution time for clarity.

2) If a command takes no arguments apart from the execution time, the second column is empty.

3) The | character separates alternative choices for an argument.

4) When an alphanumeric string is required as an argument, it must start with a letter and it must be unique during the simulation.

5) All string arguments are case sensitive as are command names.

6) When specifying a bead, bond or polymer type in a command, it is possible to use its name (e.g., H, T, etc, referred to as its *string identifier*) or its numeric type (0, 1, etc, referred to as its *numeric identifier*). Commands that expect a numeric type usually have "ByType" in their name.

7) The numeric type of beads, bonds and polymers are zero-indexed and assigned in the order that the entities appear in the input file. So the first bead type is 0, the second is 1, etc., and similarly for bonds, stiff bonds, and polymers Entities created during a simulation are given the next available numeric type at the time the command is executed.

8) If an alphanumeric string is used to name a new bead type, command target or target decorator it must be unique for the simulation. Even if the associated entity is subsequently destroyed (e.g., a

RemoveCommandTargetActivity command is used to turn off a force on a target), the name cannot be reused. Although the names strictly only have to be unique within their class (i.e., a bead name can also be used for a target, or a target name can also be used for a target decorator) this is not encouraged because of the confusion that could arise.

Monitor Commands

These commands modify the output produced by a simulation but do NOT change its execution.

Command Name	arguments	Purpose
Comment	// write some text here //	Allows comments between commands and, if the // characters are placed around other commands, it removes them from execution. Spaces must be present between text and the slash characters
SaveAmiraCurrentState		Save a snapshot in Amira format
SaveBeadDensityFluctuations	**beadName -** bead's string identifier **dataPoints** - number of values **densityPeriods** - number of periods to calculate over **x, y, z** - integer coordinates of density grid cell to analyse **bConjugate** - 0 | 1, flag showing whether to write out the fluctuations in the remainder of the simulation box as well	Write to file an array containing the number of beads of specified type in a rectangular volume over time. The x,y,z values multiply the respective *Grid* keyword values in the input file to define the rectangular space within which the fluctuations are calculated The *densityPeriods* parameter specifies the number of DensityPeriod ranges over which to average each measurement, and the dataPoints value specifies the number of values to write file.

Command Name	arguments	Purpose
SaveBeadDensityFluctuationsByType	**beadType** - bead's numeric type (0-indexed) **dataPoints** - number of values **densityPeriods** - number of periods to calculate over **x, y, z** - integer coordinates of density grid cell to analyse **bConjugate** - 0 | 1, flag showing whether to write out the fluctuations in the remainder of the simulation box as well	As previous command but uses bead's numeric type instead of its name.
SaveBead1dDensityProfile	**beadType** - bead's numeric type (0-indexed) **start** - >= 1 **end** - <= TotalTime **samplePeriod** - must divide (end+1-start) **normalVector** - (1,0,0) | (0,1,0) | (0,0,1) **sliceTotal** - integer	Calculate a 1d density function (with sliceTotal values) of a bead type along the normal direction between two times
SaveCurrentState		Save a snapshot in the current default format
SaveParaviewCurrentState		Save a snapshot in Paraview format
SavePovrayCurrentState		Save a snapshot in Povray format
SaveRestartState		Save a restart state
SetAllBeadsInvisible		Make all beads invisible
SetAllBeadsVisible		Make all beads visible
SetBeadDisplayId	**beadName** - bead string identifier **displayId -** integer >= -1	Change the seleted bead type's colour in snapshots; a value of -1 restores the colour to that set by the bead's numeric id
SetBeadTypeDisplayId	**beadType -** bead's numeric type (0-indexed) **displayId -** integer >= -1	As previous command but uses bead's numeric type instead of its name.
SetCurrentStateCamera	**xc, yc, zc** - [-inf, +inf] camera coordinates **x0, y0, z0** -[-inf, +inf] look-at point coordinates	Set the camera (xc, yc, zc) and look at points (x0, y0, z0) for Povray snapshots as multipliers of the box size. They must not be the same point.

Command Name	arguments	Purpose
SetCurrentStateDefaultFormat	**Povray | Paraview | Amira | SolventFree | SolventFreeAndPovray**	Set the default format for snapshots
SetDensityPeriod	**newPeriod**	Change the frequency of writing density fields
SetDisplayBeadRange	**axis -** x | y | z **minFraction** - [0, 1] **maxFraction** - [0, 1]	Restrict the beads in a snapshot to those in a rectangular slice of the box defined by two fractions along a major axis
SetDisplayPeriod	**newPeriod**	Change the frequency of writing snapshots
SetPolymerDisplayId	**polymerName** - polymer's string identifer **displayId** - integer >= -1	Set the same colour for all beads in the selected polymer type in snapshots; a value of -1 restores all bead types to their original colours
SetPolymerTypeDisplayId	**polymerType** - polymer's numeric type (0-indexed) **displayId** - integer >= -1	As previous command but uses numeric type of the polymer not its name
SetRestartPeriod	**newPeriod**	Change the frequency of writing restart states
ToggleBeadDisplay	**beadName**	Turn on/off display of a bead type in snapshots
ToggleDensityFieldOutput		Toggle on/off the density field output
TogglePolymerDisplay	**polymerName**	Turn on/off the display of a polymer type in snapshots

Constraint Commands

These commands change conditions in the simulation and usually modify its subsequent evolution. They are also used to create *Command Targets* that are collections of beads or polymers to which subsequent commands can be sent to carry out actions or modify them.

Command Name	arguments	Purpose
FreezeBeadsInSlice	**normalVector** - (1,0,0) | (0,1,0) | (0,0,1) **lowerBound** - [0, boxSize] **thickness** - [0, boxSize]	Freeze all beads in a rectangular slice, but they still interact with other beads
GravityOff		Turn preset gravity force off (Gravity keyword must be present in input file)
GravityOn		Turn preset gravity force on (Gravity keyword must be present in input file)
SelectBeadTypeInCylinder	**targetLabel** - unique alphanumeric string starting with a letter **beadName** - bead's string identifier **normalVector** - (1,0,0) | (0,1,0) | (0,0,1) **cx, cy, cz** - [0, 1], centre point as fraction of box size **halfLength** - [0, 0.5] **innerRadius** - units of bead diameter **outerRadius** - units of bead diameter	Create a cylindrical command target from the specified bead type. The centre point, normal vector, half length and inner and outer radius are required.
SelectBeadTypeInEllipsoid	**targetLabel** **beadName** **cx, cy, cz** - [0, 1], centre point as fraction of box size **boundingRadius** - radius of a sphere that bounds the ellipsoid **sma** - [0, 1] semi-major axis of ellipsoid **smb** - [0, 1] first semi-minor axis **smc** - [0, 1] second semi-minor axis **theta** - [0, 180] polar angle of ellipsoid's axis (deg) **phi** - [0, 360] azimuthal angle of ellipsoid's axis (deg)	Create an ellipsoidal command target from the specified bead type. The axis lengths must be in the order: sma > smb > smc
SelectBeadTypeInPentagon	**targetLabel** **beadName** **cx, cy, cz** - [0, 1], centre point as fraction of box size **boundingRadius** - radius of a sphere that bounds the pentagon (units of bead diameter) **side** - length of pentagon's side (units of bead diameter) **thickness** - depth of the pentagon (units of bead diameter) **theta** - [0, 180] polar angle of pentagon normal (deg) **phi** - [0, 360] azimuthal angle of pentagon normal (deg)	Create a pentagonal command target from the specified bead type. The pentagon's side length and thickness must be less than the bounding radius
SelectBeadTypeInSimBox	**targetLabel** **beadName**	Put all beads of the given type into a command target

Command Name	arguments	Purpose
SelectBeadTypeInSlice	**targetLabel** **beadName** **normalVector** - (1,0,0) | (0,1,0) | (0,0,1) **cx, cy, cz** - [0, 1], centre point as fraction of box size **halfX** **halfY** - [0, 0.5] half widths of slice **halfZ**	Create a rectangular slice command target from the specified bead type
SelectBeadTypeInSphere	**targetLabel** **beadName** **cx, cy, cz** - [0, 1], centre point as fraction of box size **innerRadius** - units of bead diameter **outerRadius** - units of bead diameter	Create a spherical command target from the specified bead type. If innerRadius is 0, it creates a solid sphere, if not it creates a spherical shell
SelectBeadTypeInSphericalCap	**targetLabel** **beadName** **cx, cy, cz** - [0, 1], centre point as fraction of box size **innerRadius** - units of bead diameter **outerRadius** - units of bead diameter **theta** - [0, 180] polar angle of ellipsoid's axis (deg) **phi** - [0, 360] azimuthal angle of ellipsoid's axis (deg) **gamma** - [0, 90] half-angle of cap (deg)	Create a spherical cap command target from the specified bead type
SelectPolymerTypeHeadInCylinder	**See corresponding bead command**	As for the corresponding bead command, except that it selects polymers whose head beads lie within the specified geometric region
SelectPolymerTypeHeadInEllipsoid	**See corresponding bead command**	
SelectPolymerTypeHeadInPentagon	**See corresponding bead command**	
SelectPolymerTypeHeadInSlice	**See corresponding bead command**	
SelectPolymerTypeHeadInSphere	**See corresponding bead command**	
SelectPolymerTypeHeadInSphericalCap	**See corresponding bead command**	
SelectPolymerTypeInSimBox	**See corresponding bead command**	

Command Name	arguments	Purpose
SetBondStiffness	**stiffBondName** (e.g., HHH) **bendingConstant** (>= 0) **preferredAngle** (in degrees, 0 = straight)	Modify the bending potential spring constant and preferred angle for the specified bond type by name
SetBondStrength	**bondName** (e.g., HH) **springConstant** (>= 0) **unStretchedLength** (units of bead diameter)	Modify the Hookean spring constant and the unstretched length for the specified bond type by name
SetBondStrengthbyType	**bondType** - bond's numeric type (0-indexed) **springConstant** (>= 0) **unstretchedLength** (units of bead diameter)	As previous command but uses bond's numeric type
SetDPDBeadConsInt	**firstBeadName** - string identifier **secondBeadName** - string identifier **consForceParam** (negative for an attractive force)	Modify the non-bonded conservative force parameter for the given bead types using their names
SetDPDBeadConsIntByType	**firstBeadType** (0-indexed) **secondBeadType** (0-indexed) **consForceParam** (negative for an attractive force)	As previous command but uses the beads' numeric types
SetDPDBeadDissInt	**firstBeadName** - string identifier **secondBeadName** - string identifier **dissForceParam** (>= 0)	Modify the non-bonded dissipative force parameter for the given bead types using their names
SetDPDBeadDissIntByType	**firstBeadType** - bead's numeric type (0-indexed) **secondBeadType** - bead's numeric type (0-indexed) **dissForceParam** (>= 0)	As previous command but uses the beads' numeric types
SetTimeStepSize	**stepSize** (keep in range [0.001 -0.04]	Change the integration step size
WallOff		Toggle preset wall off (Wall keyword must be present in input file)
WallOn		Toggle preset wall on (Wall keyword must be present in input file)

Target Commands

See the next section - <u>Section 9 Command Targets</u> - that describes how to create command targets to which these commands can be sent.
Target commands are directed at user-created targets, and can modify their properties or behaviour, e.g., apply an external force to a target or change the colour of the target's beads/polymers, or measure properties of the target and write the results to the log file, e.g., calculating the radius of gyration of all the beads in a target. See the description in the next section.

Note the definitions: a command target *decorator* wraps a command *target* for the purpose of carrying out an action (e.g., applying a force, counting beads, etc). An *active* command target is one that has at least one decorator currently defined. The effects of a decorator are stopped by issuing the command *RemoveCommandTargetActivity* to destroy the decorator instance.

Command Name	arguments	Purpose
AnchorForceOnTarget	**targetName** - string identifier for a target **xn, yn zn** - [-inf, +inf] arbitrary normal vector, all components must not be 0 **xc, yc zc** - [0, 1] origin from which bead distance is measured (units of box size) **magnitude** - (>= 0) magnitude of force - may be zero	Applies a force to each bead in the target in the plane defined by the normal vector with a magnitude proportional to the bead's distance from the axis
AssignExistingBeadType	**targetName** - string identifier for a target **beadName** - string identifier for an existing bead type	Changes the type of all beads in the target to the existing bead type. The beads will subsequently interact with forces for the assigned type
AxialForceOnTarget	**targetName** - string identifier for a target	
ChangeBeadType	**targetName** - string identifier for a target	Changes the numeric type of all beads in the target to the next available value, and assigns a random string as the beads' string identifier
ChangeBondPairType		Not implemented yet
ChangeBondType		Not implemented yet
ChangeNamedBeadType	**targetName** - string identifier for a target **newBeadName** - new string identifier	Changes the numeric type of beads in the target to the next available value, and assigns them the user-defined name

Command Name	arguments	Purpose
ConstantForceOnTarget	**targetName** - string identifier for a target **decName** - string identifier for this force decorator **xn, yn zn** - [-inf, +inf] direction of force, all components must not be 0 **magnitude** - [-inf, +inf] magnitude of force - may be zero	Apply a constant force to a target. The force is turned off by issuing a subsequent "RemoveCommandTargetActivity" command with the decName as the argument
CountBeadTypeInTarget.	**targetName** - string identifier for a target **beadType** - bead's numeric type identifier	Counts the number of beads of the specified type in the target
CountBeadsInTarget	**targetName** - string identifier for a target	Counts the accumulated number of beads of all types in the target
CylinderLinearForceOnTarget	**targetName** - string identifier for a target **xn, yn zn** - [-inf, +inf] arbitrary normal vector, all components must not be 0 **xc, yc zc** - [0, 1] origin of cylinder from which bead distance is measured (units of box size) **magnitude** - (>= 0) magnitude of force - may be zero	Applies a force to all beads in the target that is directed *inwardly* in the plane defined by the normal vector and whose magnitude is proportional to the distance of each bead from the cylinder's axis in the plane.It attempts to keep all beads on the cylinder's axis.
DistanceMovedByTarget	**targetName** - string identifier for a target **forceLabel** - string identifier for a force decorator **decLabel** - string identifier for this decorator **start** - start time of measurement **end** - end time of measurement	Writes out the distance moved by the target as a function of time in the specified interval. Both the total distance of all beads and the distance per bead are written.
ExternalWorkOnTarget	**targetName** - string identifier for a target **forceLabel** - string identifier for a force decorator **decLabel** - string identifier for this decorator **start** - start time of measurement **end** - end time of measurement	Writes out the work done on all beads in the target by the external force defined by *forceLabel* during the specified interval. It sums up all the F.dx elements for each bead in the target.

Command Name	arguments	Purpose
FreezeBeadsInTarget	**targetName** - string identifier for a target	Prevent all beads in the target from moving: they still interact with other beads
ListActiveCommandTargets		Writes a list of all targets currently active, i.e., that have at least 1 decorator
ListAllCommandTargetActivities		Writes a list of all decorators for all targets with the target name last on each line
ListCommandTargetActivities	**targetName** - string identifier for a target	Writes a list of all decorators wrapping the specified target
ListCommandTargets		Writes a list of all command targets whether active or not
MSDOfPolymerTarget	**targetName** - string identifier for a target **decName** - string identifier for this decorator **startTime** - start time for output (>= execution time of command) **endTime** - end time for output	Writes out the mean-square displacement of the polymers in the target during the given time interval
PlanarAnchorForceOnTarget	**targetName** - string identifier for a target **decName** - string identifier for this decorator **xn,yn,zn** - normal vector to plane **xc,yc,zc** - arbitrary point in plane **keff** - spring constant	Applies a Hookean spring force to each bead in the target from the plane defined by the normal vector/point with a magnitude proportional to the bead's distance from the axis
PolymerisePolymersInTarget	**targetName** - string identifier for a target **maxBonds** - max bonds per pair **range** - max separation of polymers **fraction** - (0,1) **springConstant** - Hookean spring constant **unstretchedLength** - Hookean spring length	Binds together a given fraction of polymers in a target with multiple Hookean springs; only polymers within speceified range are connected

Command Name	arguments	Purpose
RadialForceOnTarget	**targetName** - string identifier for a target **xn, yn zn** - [-inf, +inf] arbitrary normal vector, all components must not be 0 **xc, yc zc** - [0, 1] origin of cylinder from which bead distance is measured (units of box size) **magnitude** - (>= 0) magnitude of force - may be zero	Applies a force to all beads in the target that is directed in the plane defined by the normal vector whose magnitude is proportional to the distance of each bead from the cylinder's axis in the plane.
RemoveActiveCommandTarget	**targetName -** string identifier for a target	Removes all decorators for the named target but does not destroy the target. Further commands can be sent to it.
RemoveCommandTargetActivity	**decName -**string identifier for a target decorator	Removes the named decorator from a target. Typically used to turn off a force being applied to the target.
RgOfBeadTarget	**targetName** - string identifier for a target **decName** - string identifier for this decorator **startTime** - start time for output (>= execution time of command) **endTime** - end time for output	Writes out the radius of gyration of all beads in the target
RgOfPolymerTarget	**targetName** - string identifier for a target **decName** - string identifier for this decorator **startTime** - start time for output (>= execution time of command) **endTime** - end time for output	Writes out the radius of gyration of all polymers in the target
RotationalMSDOfPolymerTarget	**targetName** - string identifier for a target **decName** - string identifier for this decorator **startTime** - start time for output (>= execution time of command) **endTime** - end time for output	Writes out the rotational mean-square displacement of all polymers in the target
SelectBeadsInSphericalTargetRegion		
SetBondStiffnessByPositionInTarget		
SetBondStiffnessInTarget		
SetBondStrengthByPositionInTarget		
SetBondStrengthInTarget		

Command Name	arguments	Purpose
SetTargetBeadTypeDisplayId	**targetName** - string identifier for a target **beadType** - bead type **displayId** - new display id	Changes the display id (and colour) of all beads of the specified type in the target
SetTargetDisplayId	**targetName** - string identifier for a target **displayId** - new display id	Changes the display id (and colour) of all beads in the target
SetTargetInvisible	**targetName** - string identifier for a target	Set all beads in the target invisible in current state snapshots
SetTargetPolymerTypeDisplayId	**targetName** - string identifier for a target **polymerType** - polymer type **displayId** - new display id	Changes the display id (and colour) of all beads in the specified polymer type in the target
SetTargetVisible	**targetName** - string identifier for a target	Display all beads in the target in current state snapshots
SineForceOnTarget	**targetName** - string identifier for a target **decName** - string identifier for this force decorator **xn, yn zn** - [-inf, +inf] direction of force, all components must not be 0 **amplitude** - [-inf, +inf] magnitude of force - may be zero **period** - [1, +inf] period of the force	Apply a sinusoidal force to the target from the time of execution. The normal vector is normalised before applying the force. The amplitude must be positive or zero, and the period must be at least 1. The frequency is defined as 2π/period and multiplies the simulation time minus the start time.
SpringForceBetweenTargets		
SpringForceOnTarget		
ToggleAntiTargetDisplay		
ToggleTargetDisplay		
UnFreezeBeadsInTarget	**targetName** - string identifier for a target	Unfreeze the beads in the target so they can move again

9) Command Targets

It is often useful to be able to manipulate the properties of sets of beads or polymers during a simulation. This can be used to modify their appearance in snapshots to distinguish them from the environment. But a more powerful use is to change their mutual interactions or apply external forces to them, and so modify their dynamics in the simulation. Once a target has been created, its name can be used in other commands to execute actions on it. The set of commands that can be sent to a target depends on whether it is a bead or polymer target, and are given in the Target Command list above.

A *Command Target* is a set of beads or polymers that have been grouped together according to a certain criterion and given a unique label by which their properties can be modified by subsequent commands. The most common way of defining a target is to select all beads whose centres of mass lie within a specified geometric region, e.g., sphere, cylinder, planar slice, etc. This creates a *Bead Target*. Alternatively, a set of polymers whose head beads (i.e., the first bead specified in their shape string unless redefined with the Head/Tail parameters) lie within a geometric region can be selected to create a *Polymer Target*. Different commands can be sent to bead and polymer targets.

The format of all commands that create targets is similar, but the arguments differ depending on the geometric shape. Here is the command to create a spherical bead target:

```
Command SelectBeadTypeInSphere  1   targetName  beadName  xc yc zc
rin rout
```

and here is the corresponding command for a cylindrical bead target:

```
Command SelectBeadTypeInCylinder  1   targetName  beadName  xn yn
zn xc yc zc half rin rout
```

Both commands provide a unique name *targetName* for the newly-created target and the string name of the bead type to be selected - *beadName*.

The spherical target command then requires the centre of the sphere - xc, yc, zc - to be specified as a fraction of the box size, and the inner and outer radii - rin, rout - in units of the bead diameter. If rin is non-zero, a spherical shell is created.

The cylinder target command requires the normal vector along the long axis of the cylinder, which must be one of 1,0,0 or 0,1,0 or 0,0,1, the centre of the cylinder, again as a fraction of the simulation box size, and the half length of the cylinder, its inner radius, and its outer radius all in units of the bead diameter.

Further examples of geometric targets that can be created are given in the Constraint Command list above. In particular, all the beads or polymers of a single type in the simulation can be selected with the commands **SelectBeadTypeInSimBox** or **SelectPolymerTypeInSimBox**.

The following four commands illustrate a typical use of a bead target. They all execute at time 25000 and create a bead target and then change its colour in snapshots and modify its non-bonded interaction with another bead type.

The first command creates a bead target called **bolus** that contains all beads of type **W** that lie in a sphere with its centre in the middle of the simulation box (at 0.5, 0.5, 0.5), and an inner and outer radius of 0 and 4.0.

The second command changes the display type of these beads so that they appear as a different colour in snapshots. Note that this does not change their interactions: the display id is just a paramete

The third command changes the type of the beads to a new value with the new name fluors. This allows them to be the target of subsequent commands by using their new name or numeric type. The fourth command change the conservative force parameter for interactions of these beads with W beads to 35.

```
Command  SelectBeadTypeInSphere 25000  bolus W 0.5 0.5 0.5 0.0 4.0

Command  SetTargetDisplayId        25000 bolus   4

Command  ChangeNamedBeadType       25000 bolus   fluors

Command  SetDPDBeadConsInt         25000 fluors  W  35
```

However, even though the beads have had their properties changed, they are still moving independently and will diffuse away from each other over time. To make the target rigid so that it moves as a single object, we have to create a second target that contains the polymers corresponding to these beads, and then tie them together using newly-created bonds. For this, we create a polymer target that exactly overlaps the same volume.

For this example, we assume that polymers of type Water contain the single bead W.

```
SelectPolymerTypeHeadInSphere  25000 bolusPoly Water  0.5 0.5 0.5
0.0 4.0

PolymerisePolymersInTarget      25000 bolusPoly  12 1.5 1.0 128.0
0.5
```

Whereas the bead target required a bead name, the polymer target requires the corresponding polymer name. The geometric parameters are the same as before. The second command then creates Hookean spring bonds between the head beads in the polymers in the bolusPoly target. The arguments are, in order:

12   = the maximum number of bonds created per pair of polymers
1.5   = maximum range out to which two beads will have a bond created between them
1.0   = the fraction of polymers that will be bonded; fractions less than 1 create floppy target
128.0 = the Hookean spring constant
0.5   = the Hookean spring unstretched length

10) Analysing the results of a simulation

Two types of analysis are performed automatically in all simulations: time-averaged data, which are written to the **Analysis State file,** and time-series data, which are written to the **History State file.**

All commands that are specified in the input file and that execute correctly write a message to the **Log State file - dmpcls.nnn**. This file also contains the results of some user-specified analysis that is turned on by command. See the appropriate commands in <u>Section 8</u> above. If a command fails to execute (e..g, if its name is misspelt or its arguments are incorrect), a warning or error message is written to the log file.

The **Restart States - dmpcrs.nnn.con.ttt.dat -** contain information on all beads and polymers and their position, momenta, and forces in ascii text format that is required to restart a simulations. Offline analysis can be performed on the contents of these files as they contain plain text.

Analysis State File - dmpcas.nnn

Observables are written here every AnalysisPeriod time steps during a simulation, and all data are averaged over AnalysisPeriod / SamplePeriod values. Each AnalysisPeriod number of time steps, observables are averaged over all samples taken since the last analysis was performed, and written to this file.

Each block of data starts with the simulation time, temperature and pressure, and is followed by other observables:

```
Time = 100000
Temperature
    1.0080152    0.046555619

Pressure
    23.352143    0.11033159
```

Scalar observables are presented as Mean / Standard deviation pairs on the same line.

When certain special analysis options are switched on, extra data will be written to this file. Ask me for more details.

Because a lot of data is present in this file, we do not describe it all here. But the following observables are always calculated:

Temperature

Pressure

Centre of mass momentum of all beads (this should be zero as the CM of the simulation box should not be moving), and position (should be the middle of the simulation box)

Stress tensor (3 x 3 matrix) of all beads, and **spherical stress tensor**

Inertia tensor of all beads

Bond length of all bead types combined

Sequence of bond lengths for all types of bond defined in the simulation

Sequence of end-to-end lengths for all polymer types defined (polymers that contain only a single bead have an end-to-end length of zero)

Angular and bond length measures of the stiff bond types defined

History State File - dmpchs.nnn

The history state file contains time series of various observables. They are written out every SamplePeriod number of time steps into the following columns:

Time

Temperature

Pressure

(next are 3 columns of zeroes for observables not used in DPD)

Set of bead diffusion constants for all bead types defined in the input file

Set of polymer end-to-end lengths for all polymer types defined in the input file

Note that even if a bead type is not used in any polymer, and so no instances are created, it will still have a column in the history file that will contain all zeroes. Also note that if a new bead type is created as the result of a command during a simulation, an extra column for the bead's diffusion constant will be added to this file from the command's execution time onwards.

Conditionally-created analysis files

When specialised analysis options are used, extra files are created to hold the data. See me for more details.

---

11) Overview of Dissipative particle dynamics

Molecular Dynamics (MD) and Dissipative Particle Dynamics (DPD) are two simulation techniques that integrate Newton's laws of motion for a set of particles interacting via specified forces and generate trajectories from which the observable properties of the set can be estimated. The techniques differ in their specification of the force laws, but are otherwise quite similar. MD aims to model the inter-atomic potentials as accurately as possible, and so generate detailed information on the molecular interactions of complex systems such as proteins and lipids in aqueous solution. By contrast, DPD ignores the atomic-level details of molecules, and uses a coarse-grained set of force laws that are chosen to produce the correct hydrodynamic behaviour of fluids. The forces in DPD are all short-ranged, pairwise additive, conserve linear momentum, and have no hard- core repulsion at zero separation: thus, two particles can be at exactly the same place in space, although this is unlikely if the conservative force parameter is non-zero. This feature makes DPD especially

suitable for fluid simulations, as the representation of the strong repulsion present in the solid phase is problematic.

The elementary units in a DPD simulation are fluid elements or *beads*. A bead represents a volume of fluid that is large on a molecular scale, and hence contains at least several molecules of the fluid, but still macroscopically small. Beads interact via effective forces chosen so as to reproduce the hydrodynamic behaviour of the fluid without reference to its molecular structure. DPD differs in this respect from MD simulations, in which the forces are chosen to model the inter-molecular interactions of a system as accurately as possible. Forces in DPD are pairwise additive, conserve momentum, have no hard core and are short-ranged, the range of the force defining the size of the beads.

All beads have the same mass, m_0, and diameter, d_0, and these set the mass and length scales in the simulation. A time-scale must be extracted from the dynamics of relevant processes in the simulated fluid, such as the diffusion of a micelle's centre of mass, or the in-plane viscosity of a bilayer membrane. For example, when we study equilibrium properties of the bilayers, we use the generic time- scale set by the system temperature, $t_0 = \sqrt{(m_0 * d_0 * d_0 / k_B T)}$, where k_B is Boltzmann's constant and the temperature, T, is the mean kinetic energy of all beads.

All beads interact via three forces: a *Conservative* force that gives each bead an identity and allows, for example, the representation of hydrophobicity between hydrocarbon and water; a *Random* force that creates relative momentum between bead pairs; and a *Dissipative* force that destroys relative momentum. Beads are considered to have (unobserved) internal degrees of freedom that give rise to the dissipative force, and to be coupled to the local temperature of their (fluid) environment that is the source of the random forces. It has been shown that choosing the random and dissipative forces appropriately leads to equilibrium states of the system that satisfy the Boltzmann distribution. An important point about the random forces is that they are pairwise anti-symmetric. If one bead in an interacting pair gains an amount of relative momentum, its partner loses the same amount. This distinguishes DPD from Brownian Dynamics in which each particle receives a random push independently of all other particles.

Because DPD beads represent a volume of fluid, and not single molecules, the interpretation of a polymer composed of such beads requires some care. We take the view that the head bead in a model lipid represents the hydrophilic glycerol-phosphate-head region while each tail bead represents several methyl groups, or a Kuhn length, in a hydrocarbon chain. In this view, each hydrophobic bead represents, say, 3 to 4 methyl groups. The same interpretation applies to non-biological amphiphiles, such as sulphonium surfactants that consist of an 18-carbon chain attached to a sulphonium group and two hydroxy groups.