

# GOMC

User Manual  
Version 2.00

Distributed by the Potoff and Schwiebert Groups  
©Wayne State University

May 14, 2017

# Contents

# 1 Tutorial Overview

This document will instruct a new user how to download, compile, and run the GOMC molecular simulation code. A basic understanding of statistical physics is recommended to complete this tutorial.

To demonstrate the capabilities of the code, the user is guided through the process of downloading and compiling a GOMC executable. That executable is then used to perform saturated vapor and liquid equilibria (VLE) studies on systems of pure isobutane (R600a), a branched alkane that whose application as a refrigerant/propellant is increasing.

<http://en.wikipedia.org/wiki/Isobutane>

The Transferable Potentials for Phase Equilibria (TraPPE) united atom (UA) force field is used to describe the molecular geometry constraints and the intermolecular interactions.

## 2 Introduction

Monte Carlo (MC) simulation is a type of simulation driven by stochastic processes. "GO" stands for GPU-Optimized; this code was intended to run optimally on modern graphics process hardware.

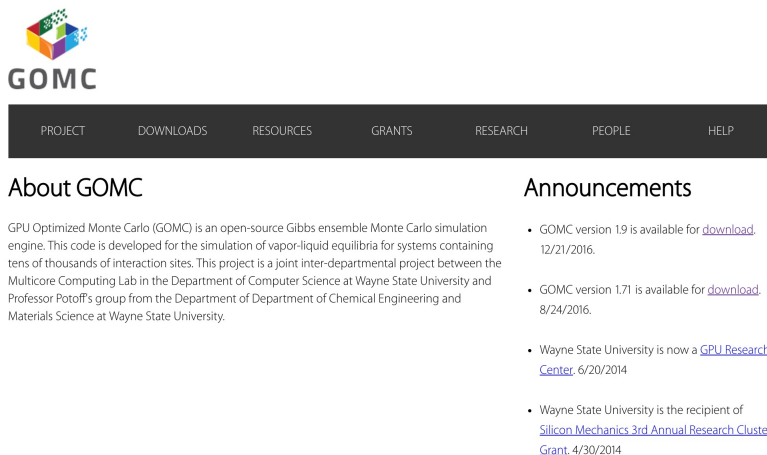
More specifically, this engine includes serial and GPU-Optimized (multi-threaded) codes designed to run Markov chain Boltzmann sampling of chemical systems – effectively sets of points defined by topological maps and interaction algorithms in a simulation box. From statistical mechanics, we know this is one way to sample phase space and model chemical systems.

GOMC currently supports canonical, isobaric-isothermal, Grand canonical, constant volume Gibbs ensemble, and constant pressure Gibbs ensemble simulations. GOMC employs widely-used simulation file types (PDB, CHARMM-style parameter file, PSF). GOMC includes configurational bias algorithms for both linear and branched charged, and none charged systems.

## 3 How to get the software

The latest public code builds, project logo, manual, and other resources can be obtained via the following website:

<http://gomc.eng.wayne.edu/>



**GOMC**

PROJECT DOWNLOADS RESOURCES GRANTS RESEARCH PEOPLE HELP

### About GOMC

GPU Optimized Monte Carlo (GOMC) is an open-source Gibbs ensemble Monte Carlo simulation engine. This code is developed for the simulation of vapor-liquid equilibria for systems containing tens of thousands of interaction sites. This project is a joint inter-departmental project between the Multicore Computing Lab in the Department of Computer Science at Wayne State University and Professor Potoff's group from the Department of Chemical Engineering and Materials Science at Wayne State University.

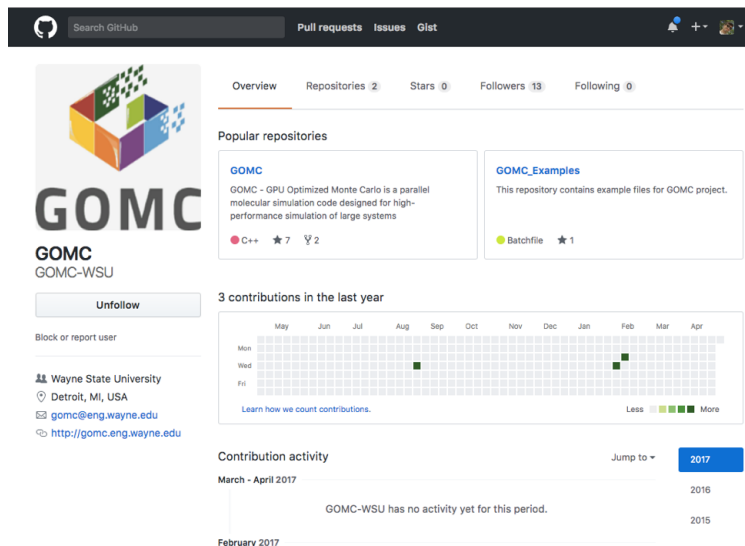
### Announcements

- GOMC version 1.9 is available for [download](#). 12/21/2016.
- GOMC version 1.71 is available for [download](#). 8/24/2016.
- Wayne State University is now a [GPU Research Center](#). 6/20/2014.
- Wayne State University is the recipient of [Silicon Mechanics 3rd Annual Research Cluster Grant](#). 4/30/2014.

The code can be found under the download tab, below and to the right of the logo. When new betas (or release builds) are announced, they will replace the prior code under the downloads tab. An announcement will be posted on the front page to notify users.

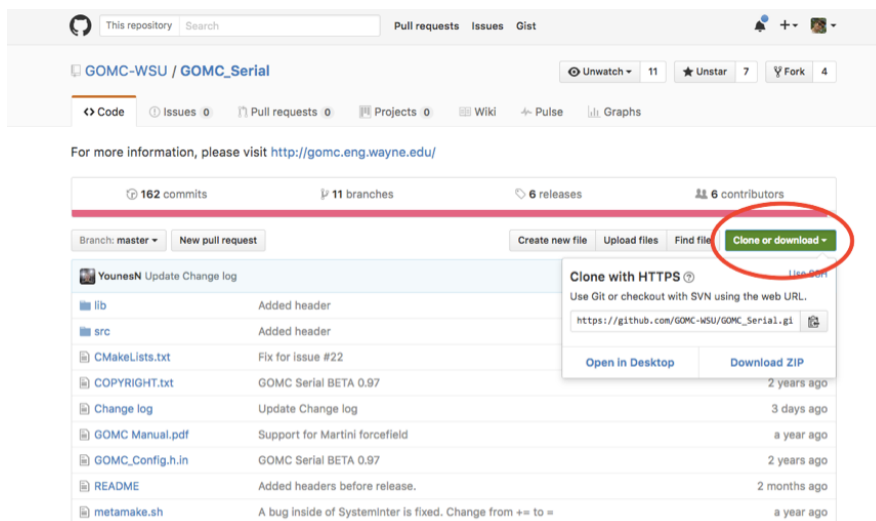
Currently, version control is handled through the GitHub repository. The posted builds in Master branch are “frozen” versions of the code that have been validated for a number of systems and ensembles. Other branches are created as a means of implementing new features. The latest updated code builds, project logo, manual, example files, and other resources can be obtained via the following GitHub repository:

<https://github.com/GOMC-WSU>



The CPU and GPU code are merged together under GOMC repository and can be found under the main page. In addition, Examples repository can be found under the main page. Under each repository, the code and manual can be downloaded by clicking on the Clone or download tab. For more information regarding GitHub, visit the following link:

<https://guides.github.com/activities/hello-world/>



## 4 Platform and Software Requirements

### 4.1 Supported Operating Systems

GOMC officially supports Windows 7, 8, and most modern distributions of Linux (see the next section). This software has the ability to compile on recent versions of OS X; however, such a platform is not officially supported.

### 4.2 Required Software Prerequisites

GOMC has some mild software requirements, which are widely available for Linux operating systems. Required software requirements are:

#### 1 C++03 Compliant Compiler

##### 1.1 Linux/OS X

##### 1.1.1 `icc` (Intel C++ Compiler)

Type the following command in a terminal:

```
$ icc --version
```

If gives a version number 4.4 or later, you're all set. If it's older than 4.4 (released in 2009), we recommend upgrading.

In Linux, the Intel compiler will generally produce the fastest serial executables (when running on Intel Core processors).

##### 1.1.2 `g++` (GNU GCC)

Type the following command in a terminal.

```
$ g++ --version
```

If gives a version number 4.4 or later, you're all set. If it's older than 4.4 (released in 2009), we recommend upgrading.

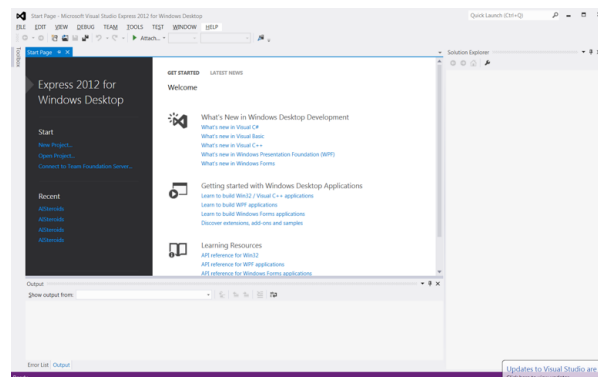
##### 1.2 Windows

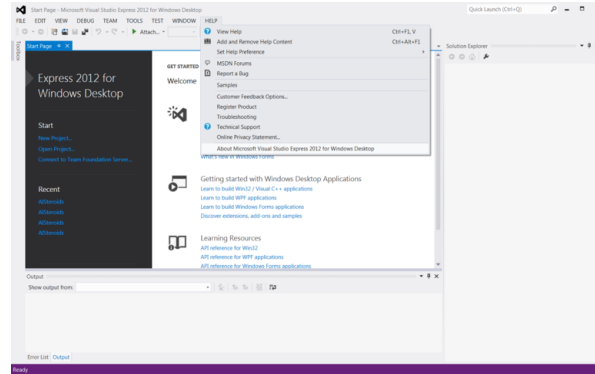
##### 1.2.1 Visual Studio

Microsoft's Visual Studio 2010 or later is recommended.

To check the version:

*Help* (top tab) → *About Microsoft Visual Studio*





- 1.2.2 cmake (if compiling on Linux)  
To check if cmake is installed:

```
$ which cmake
```

To check the version number:

```
$ cmake --version
```

- 1.2.3 nvcc/CUDA libs  
The GPU builds of the code requires NVIDIA's CUDA 6.0 or newer:  
To check if nvcc is installed:

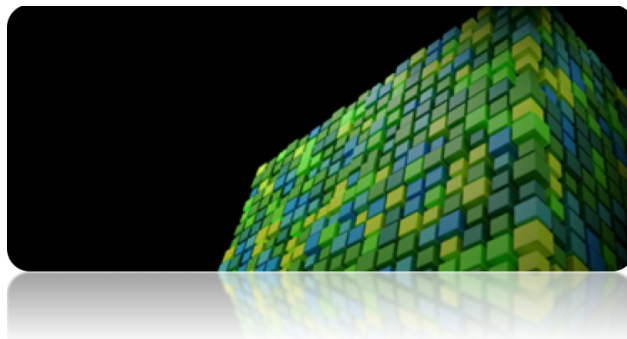
```
$ which nvcc
```

To check the version number:

```
$ nvcc --version
```

CUDA is viewed as an essential requirement, but is not used to compile the serial code, which can be compiled on systems without CUDA.  
To download CUDA visit NVIDIA's webpage:

<https://developer.nvidia.com/cuda-downloads>



CUDA is required to compile the GPU executable in both Windows and Linux. Please refer to CUDA Developer webpages to select an appropriate version for the desired platform.

To install CUDA in Linux root/sudo, privileges are generally required. In Windows, administrative access is required.

## 5 Highly Recommended Software Tools

**NOTE:** *The listed programs are used in this manual and are generally considered necessary.*

### 5.1 VMD

VMD (Visual Molecular Dynamics) is a 3-D visualization and manipulation engine for molecular systems written in C-language. VMD is distributed and maintained by the University of Illinois at Urbana-Champaign. Its sources and binaries are free to download. It comes with a robust scripting engine, which is capable of running python and tcl scripts. More info can be found here:

<http://www.ks.uiuc.edu/Research/vmd/>

Although GOMC uses the same fundamental file types ? PDB (coordinates) and PSF (topology) as VMD, it uses some special tricks to obey certain rules of those file formats. One useful purpose of VMD is visualization of your systems.

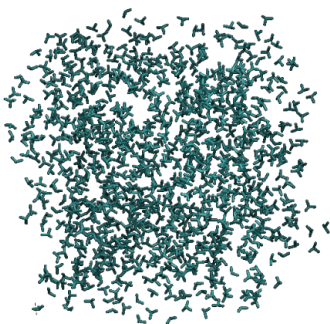


Figure 1: A system of united atom isobutane molecules

Nonetheless, the most critical part of VMD is a tool called PSFGen. PSFGen uses a tcl or python script to generate a PDB and PSF file for a system of one or more molecules. It is, perhaps, the most convenient way to generate a compliant PSF file.

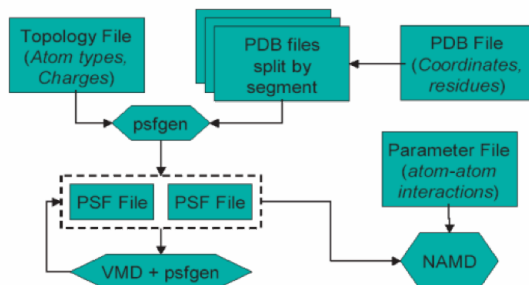


Figure 2: An overview of the PSFGen file generation process and its relationship to VMD/NAMD

To read more about PSFGen, reference:

Plugin homepage @ UIUC

<http://www.ks.uiuc.edu/Research/vmd/plugins/psfgen>

“Generating a Protein Structure File (PSF)”, part of the NAMD Tutorial from UIUC

<http://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-html/node6.html>

In-Depth Overview [PDF]

<http://www.ks.uiuc.edu/Research/vmd/plugins/psfgen/ug.pdf>

## 5.2 Packmol

Packmol is a molecule packing tool created by José Mario Martínez, a professor of mathematics at the State University of Campinas, Brazil. It is written in Fortran and is free to download. More information is available on their homepage:

<http://www.ime.unicamp.br/~martinez/packmol>

To compile it, a Fortran language compiler is needed, such as gfortran. Many Linux distributions no longer come with Fortran compilers automatically, so this may need to be installed additionally.

Packmol allows a specified number of molecules to be packed at defined separating distances within a certain region of space. One of Packmol's limitations is that it is unaware of topology; it treats each molecule or group of molecules as a rigid set of points.

**WARNING:** *Another more serious limitation is that it is not aware of periodic boundary conditions (PBC). As a result, when using Packmol to pack PDBs for GOMC, it is recommended to pack to a box 1 Angstroms smaller than the simulation box size. This prevents hard overlaps over the periodic boundary.*

## 6 Other Useful Software Tools

### 6.1 Grace

Grace is a piece of graphing software written and maintained by the Weizmann Institute of Science's Plasma Laboratory (Rehovot, Israel). Mostly used in Linux, it can also be compiled in Windows. The developers warn it may be missing some functionality.

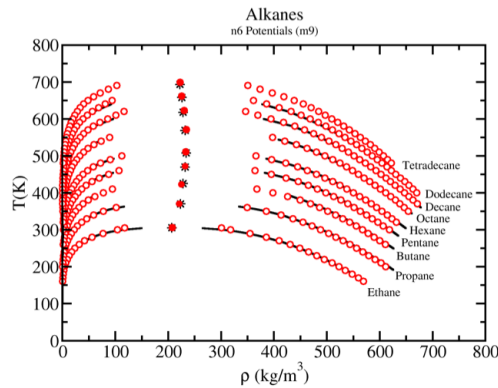


In-depth information and the source can be found on the project page, here:

<http://plasma-gate.weizmann.ac.il/Grace>

When compiled, Grace's executable in Linux is typically named “xmgrace”. This tool allows the production of high quality, precise line and dot graphs, ideal for visualizing much of the thermodynamic data from the GOMC engine. Below is an example of the results of simulations of saturated VLE densities of linear alkanes produced with Grace.

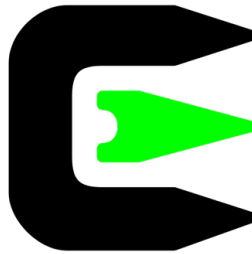




## 6.2 Cygwin

Cygwin is one option to assist in building and visualizing systems in Windows. It provides Microsoft Windows users with a Unix-like environment and command-line interface, and offers Windows-compatible ports of common Linux applications.

<https://cygwin.com>



The software is a free and open source, licensed under the GNU General Public License version 3. Its primary maintainers are Red Hat Inc. and NetApp. One of the most impressive abilities of Cygwin is its ability to launch a full Windows-compatible X-server Window, which allows convenient visualization of Linux app GUIs. It is compatible with the Grace graphing software. In practice, this package behaves most analogously to a Linux virtual machine in Windows.

## 7 Compiling GOMC

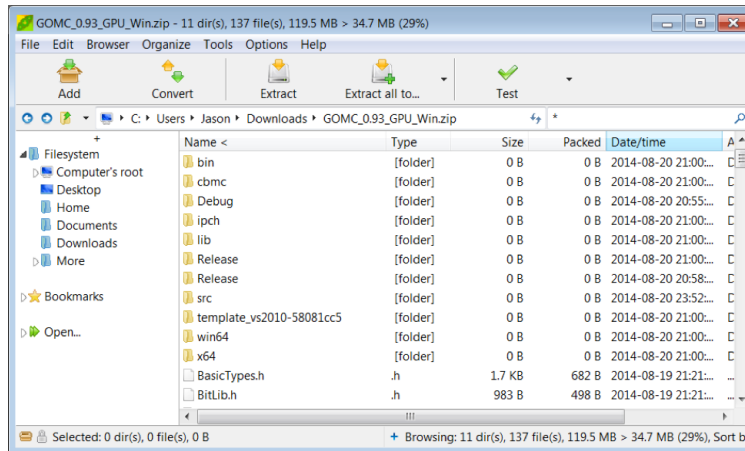
### 7.1 Extracting the code

GOMC is distributed as a compressed folder, containing the source and build system. To compile the code after downloading it, the first step is to extract the compressed build folder.

In Windows, the folder for the GPU code is compressed using a standard \*.zip file format. To unzip simply use a utility like Peazip:

<http://peazip.sourceforge.net/>

Below is an example of what the downloaded code looks like when unzipping in Peazip.



In Linux, the GPU and Serial codes are compressed using gzip and tar (\*.tar.gz). To extract, simply move to the desired folder and type in the command line:

```
$ tar -xzf <file name>.tar.gz
```

## 7.2 Compiling the code

### 7.2.1 GPU code

#### Compilation on Windows

Once the code is extracted, to compile it on Windows, you need to load the project into Visual Studio by opening the extracted folder and double clicking on the solution file of the desired Visual Studio version. After the solution is opened in Visual Studio, go to the “Build” menu and select “Build solution” to compile the code. You can compile either with release mode or with debug mode by selecting the desired mode from the “Solution Configuration” drop box. To run the project, simply click the run button or hit F5 on the keyboard.

#### Compilation on Linux

To compile the GPU code on Linux, go to the directory of the project, and type in the command line:

```
$ make
```

You can configure the “makefile” file to choose different C compilers, to select the desired compute capability, and to configure many more compilation flags.

The default compute capability is 3.0. To change the compute capability, go to the `GENCODE_FLAGS` option, and set it to one of the compute capability flags that are defined in the file.

```
# CUDA code generation flags
GENCODE_SM10    := -gencode arch=compute_10,code=sm_10
GENCODE_SM20    := -gencode arch=compute_20,code=sm_20
GENCODE_SM30    := -gencode arch=compute_30,code=sm_30
GENCODE_SM35    := -gencode arch=compute_35,code=sm_35
GENCODE_FLAGS    := $(GENCODE_SM30)
```

To run the program, run the executable “GOMC.out”. The system’s `LD_LIBRARY_PATH` will need to be configured to support CUDA (more on this later).

### 7.2.2 Serial Code

#### Compilation on Windows

See GPU “Compilation on Windows” section and follow an identical procedure for the released serial code. See README for instructions on how to use the CMake-GUI to build the configuration and solution files necessary for the Windows build.

#### Compilation on Linux

In Linux, the CPU code uses a simple makefile. Enter the directory and type in the command line:

```
$ make all
```

This will use the Makefile to compile a GPU-compatible executable called “GOMC.out”. To run, the system’s LD\_LIBRARY\_PATH will need to be configured to support CUDA (more on this later). For the serial code, which uses cmake for compilation, go to the base directory and type in the command line:

```
$ ./metamake.sh
```

This cmake script will create a directory named “bin”. Enter this directory:

```
$ cd bin
```

and type:

```
$ make
```

Four executables - GOMC.Serial.GEMC (Gibbs ensemble), GOMC.Serial.NVT (NVT ensemble), GOMC.Serial.NPT (isobaric-isothermal ensemble), and GOMC.Serial.GCMC (Grand canonical ensemble) - will be produced. By default, the distribution compiles in release mode. To compile in debug mode (if you’re using the code as a developer), open the file “CMakeCache.txt” while still in the “bin” folder. This file contains information used by cmake to build the executables. To compile in debug mode, change the value after “CMAKE\_BUILD\_TYPE:STRING=” from “Release” to “Debug”, and retype the command:

```
$ make
```

The output executables should now be compiled with debugger symbols. You can also swap the compiler by modifying the “CMAKE\_CXX\_COMPILER” variable. For more information, refer to the CMake documentation. Running GOMC in parallel using OpenMP:

To run the parallel version of CPU code, it needs to be compiled with openmp library. Open the file “CMakeCache.txt”, while still in the “bin” folder, and change the value after “CMAKE\_CXX\_FLAGS\_RELEASE:STRING=” from “-O3 -DNDEBUG” to “-O3 -qopenmp -DNDEBUG”.

And retype the command:

```
$ make
```

## 8 Input File Formats

In order to run simulation in GOMC, the following files need to be provided:

- GOMC executable
- Input file “NAME.conf” (proprietary control file)

- PDB file(s)
- PSF file(s)
- Parameter file

## 8.1 PDB File

The PDB file stores coordinates for the simulation. The file format is widely adopted.

- Protein Databank (PDB) Files (plural: PDB files)
- Open format, well-documented
- Fixed-width format (hence white space is significant)
- Up to 13.5m page views a month; up to 55.8m FTP requests per month
- Used by NAMD, GROMACS, CHARMM, ACEMD, Amber

An overview of the PDB standard can be found here:

<http://www.wwpdb.org/docs.html>

The advantage of PDB files is their ubiquity and thorough documentation. Disadvantages include limited fixed point floating precision for coordinates, unused space, and proprietary implementations creating inconsistencies.

One PDB file is required per box. For NVT ensemble simulations, one file is expected; for Gibbs and grand canonical ensemble, two files are required. GOMC recognizes the following keywords in PDB files:

- ◇ REMARK
- ◇ CRYST1
- ◇ ATOM
- ◇ END

Currently, REMARK is ignored. Formerly, it was used to store proprietary information in frames (e.g. step number). Packmol typically leaves the following remark:

REMARK	original	generated	coordinate	pdb	file
--------	----------	-----------	------------	-----	------

at the top of the file. Note that this is another example of an inconsistency with the spec. As of the PDB v3.30 specification the REMARK entry contains an identifying integer, which is supposed to occupy lines 8-10.

## REMARKS

### Overview

REMARK records present experimental details, annotations, comments, and information not included in other records. In a number of cases, REMARKs are used to expand the contents of other record types. A new level of structure is being used for some REMARK records. This is expected to facilitate searching and will assist in the conversion to a relational database.

The very first line of every set of REMARK records is used as a spacer to aid in reading.

COLUMNS	DATA TYPE	FIELD	DEFINITION
1 - 6	Record name	"REMARK"	
8 - 10	Integer	remarkNum	Remark number. It is not an error for remark n to exist in an entry when remark n-1 does not.
12 - 79	LString	empty	Left as white space in first line of each new remark.

REMARK 3  
REMARK 0,1,2,4,5-299  
REMARK 300-999

A file generated by Packmol has "ori" in this position. Hence you may see future codes that are incompatible with this legacy kind of remarks.

Note also that the spaces 7 and 11 are not reserved; hence, they may be used in proprietary specifications. CRYST1 can be used to store the cell dimensions, which can also be put as a tag in the proprietary control file.

<http://www.wwpdb.org/documentation/format33/sect8.html#CRYST1>

## Crystallographic and Coordinate Transformation Section

This section describes the geometry of the crystallographic experiment and the coordinate system transformations.

### CRYST1

#### Overview

The CRYST1 record presents the unit cell parameters, space group, and Z value. If the structure was not determined by crystallographic means, CRYST1 simply provides the unitary values, with an appropriate REMARK.

#### Record Format

COLUMNS	DATA	TYPE	FIELD	DEFINITION
1 - 6	Record name		"CRYST1"	
7 - 15	Real (9.3)		a	a (Angstroms).
16 - 24	Real (9.3)		b	b (Angstroms).
25 - 33	Real (9.3)		c	c (Angstroms).
34 - 40	Real (7.2)		alpha	alpha (degrees).
41 - 47	Real (7.2)		beta	beta (degrees).
48 - 54	Real (7.2)		gamma	gamma (degrees).
56 - 66	LString		sGroup	Space group.
67 - 70	Integer		z	Z value.

#### Details

- If the entry describes a structure determined by a technique other than X-ray crystallography, CRYST1 contains  $a = b = c = 1.0$ ,  $\alpha = \beta = \gamma = 90$  degrees, space group = P 1, and  $Z = 1$ .
- The Hermann-Mauguin space group symbol is given without parenthesis, e.g., P 43 21 2. Please note that the screw axis is described as a two digit number.
- The full International Table's Hermann-Mauguin symbol is used, e.g., P 1 21 1 instead of P 21.
- For a rhombohedral space group in the hexagonal setting, the lattice type symbol used is H.
- The Z value is the number of polymeric chains in a unit cell. In the case of heteropolymers, Z is the number of occurrences of the most populous chain.

As an example, given two chains A and B, each with a different sequence, and the space group P 2 that has two equipoints in the standard unit cell, the following table gives the correct Z value.

**NOTE:** Only cubic and orthogonal cells are supported in this code. The main entry in the PDB file are ATOM| entries. The keyword "ATOM" is always followed by two spaces. An entry has a number of fields.

## Coordinate Section

The Coordinate Section contains the collection of atomic coordinates as well as the MODEL and ENDMDL records.

### ATOM

#### Overview

The ATOM records present the atomic coordinates for standard amino acids and nucleotides. They also present the occupancy and temperature factor for each atom. Non-polymer chemical coordinates use the HETATM record type. The element symbol is always present on each ATOM record; charge is optional.

Changes in ATOM/HETATM records result from the standardization atom and residue nomenclature. This nomenclature is described in the Chemical Component Dictionary (<ftp://ftp.wwpdb.org/pub/pdb/data/monomers>).

#### Record Format

COLUMNS	DATA	TYPE	FIELD	DEFINITION
1 - 6	Record name	"ATOM "		
7 - 11	Integer	serial		Atom serial number.
13 - 16	Atom	name		Atom name.
17	Character	altLoc		Alternate location indicator.
18 - 20	Residue name	resName		Residue name.
22	Character	chainID		Chain identifier.
23 - 26	Integer	resSeq		Residue sequence number.
27	AChar	iCode		Code for insertion of residues.
31 - 38	Real (8.3)	x		Orthogonal coordinates for X in Angstroms.
39 - 46	Real (8.3)	y		Orthogonal coordinates for Y in Angstroms.
47 - 54	Real (8.3)	z		Orthogonal coordinates for Z in Angstroms.
55 - 60	Real (6.2)	occupancy		Occupancy.
61 - 66	Real (6.2)	tempFactor		Temperature factor.
77 - 78	LString(2)	element		Element symbol, right-justified.
79 - 80	LString(2)	charge		Charge on the atom.

#### Details

- ATOM records for proteins are listed from amino to carboxyl terminus.
- Nucleic acid residues are listed from the 5' to the 3' terminus.
- Alignment of one-letter atom name such as C starts at column 14, while two-letter atom name such as FE starts at column 13.
- Atom nomenclature begins with atom type.
- No ordering is specified for polysaccharides.
- Non-blank alphanumeric character is used for chain identifier.
- The list of ATOM records in a chain is terminated by a TER record.
- If more than one model is present in the entry, each model is delimited by MODEL and ENDMDL records.
- AltLoc is the place holder to indicate alternate conformation. The alternate conformation can be in the entire polymer chain, or several residues or partial residue (several atoms within one residue). If an atom is provided in more than one position, then a non-blank alternate location indicator must be used for each of the atomic positions. Within a residue, all atoms that are associated with each other in a given conformation are assigned the same alternate position indicator. There are two ways of representing alternate conformation- either at atom level or at residue level (see examples).
- For atoms that are in alternate sites indicated by the alternate site indicator, sorting of atoms in the ATOM/HETATM list uses the following general rules:
  - In the simple case that involves a few atoms or a few residues with alternate sites, the coordinates occur one after the other in the entry.
  - In the case of a large heterogen groups which are disordered, the atoms for each conformer are listed together.
- Alphabet letters are commonly used for insertion code. The insertion code is used when two residues have the same numbering. The combination of residue numbering and insertion code defines the unique residue.
- If the depositor provides the data, then the isotropic B value is given for the temperature factor.
- If there are neither isotropic B values from the depositor, nor anisotropic temperature factors in ANISOU, then the default value of 0.0 is used for the temperature factor.
- Columns 79 - 80 indicate any charge on the atom, e.g., 2+, 1-. In most cases, these are blank.
- For refinements with program REFMAC prior 5.5.0042 which use TLS refinement, the values of B may include only the TLS contribution to the isotropic temperature factor rather than the full isotropic value.

The key parameters are the coordinates x, y, and z. The precision is limited to eight whole decimal digits and three fractional decimal digits.

Other important entries are the residue name, atom name, and chain ID. Numbering is important primarily because it represents an inconvenience in packing/loading large systems. Revisiting the previous example,

the atom name is “C1” and residue name is “ISB”. The PSF file (next section) contains a lookup table of atoms. These contain the atom name from the PDB and the name of the atom kind in the parameter file it corresponds to. As multiple different atom names will all correspond to the same parameter, these can be viewed “atom aliases” of sorts. The chain letter (in this case ‘A’) is sometimes used when packing a number of PDBs into a single PDB file.

A few important **Notes** / **Warnings** on Undocumented PDB Format Conventions:

- While it is explicitly stated in some other sections of the PDB file, the general convention observed by most codes is to right align when padding with white space.
- Some codes (including PSFGen/VMD) use the 21st unused character to add a fourth letter to the residue (molecule name). This extension is currently supported, but is unofficial and, hence, may change in the future.
- VMD requires a constant number of ATOMS in a multi-frame PDB (multiple records terminated by “END” in a single file). To compensate for this, all atoms from all boxes in the system are written to the output PDBs of this code.
- For atoms not currently in a box, the coordinates are set to  $< 0.00, 0.00, 0.00 >$
- The occupancy is commonly just set to “1.00” and is left unused by many codes. We recycle this legacy parameter by using it to denote, in our output PDBs, the box a particle is in (box 0 occupancy=0.00 ; box 1 occupancy=1.00)
- As the x, y, and z coordinates are fixed point with only three digits of precision, the energy values you get when restarting may be mildly different, particularly for bonded interactions due to roundoff in the coordinates. This will eventually be remedied by the implementation of a full-precision trajectory (e.g. DCD) file.
- The “ISB” entry in columns 73-75 is not an official part of the PDB standard. This is a proprietary entry called “Segname”, which has been embraced by NAMD and some other codes.

A frame in the PDB file is terminated with the keyword **END**.

With that overview of the format in mind, the following steps describe how a PDB file is typically built.

1. A single molecule PDB is obtained. In this example, the QM software package Gaussian was used to draw the molecule, which was then edited by hand to adhere to the PDB spec properly. The end result is a PDB for a single molecule:

```
REMARK      1 File created by GaussView 5.0.8
ATOM        1  C1  ISB  1  0.911  -0.313  0.000  C
ATOM        2  C2  ISB  1  1.424  -1.765  0.000  C
ATOM        3  C3  ISB  1  -0.629  -0.313  0.000  C
ATOM        4  C4  ISB  1  1.424   0.413  -1.257  C
END
```

2. Next, packings are calculated to place the simulation in a region of vapor-liquid coexistence. There are a couple of ways to do this in Gibbs ensemble:
  - Pack both boxes to a single “middle” density, which is an average of the liquid and vapor densities.
  - Same as 1, but add a modest amount to axis of one box (e.g. 10-30 Å). This technique can be handy in the constant pressure Gibbs ensemble.



[illegible]

A good reference for getting the information needed to estimate packing is the NIST Web Book database of pure compounds:

<http://webbook.nist.gov/chemistry/>

3. After packing is determined, a basic pack can be performed with a Packmol script. Here is one example:

```
tolerance 3.0
filetype pdb
output STEP2_ISB_packed_BOX_0.pdb

structure isobutane.pdb
number 1000
inside box 0.1 0.1 0.1 70.20 70.20 70.20
end structure
```

Packmol scripts are typically saved with the extension `*.inp`, so this might be named `"pack_isobutane.inp"`. To run the script, we type the following line into the terminal:

```
$ ./packmol < pack_isobutane.inp
```

## 8.2 PSF File

The PSF file stores the topology, mass, charges, and atom identities of molecules in the system.

- Protein Structure File (PSF)
- Space-separated file
- Used by NAMD, CHARMM, X-PLOR

The PSF file is not as robustly documented as the PDB format, but a basic description of it can be found here:

<http://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-win-html/node24.html>

The PSF file is generally composed of a series of sections. A line with a numeric value is typically at the top of each section. This value lists the number of entries in that section (lines can contain multiple entries; a dihedral, for example has two quadruplet entries of atom indices per line). Note that outside the remarks and atom section, this number is typically smaller than the number of lines by a factor of 2 to 4.

PSF files always start with the string “PSF” on their first line.

GOMC reuses PSF reading code from NAMD, hence it should have much of the same flexibility and limitations. By section, the segments of a PSF file are:

- TITLE: remarks on the file
- BONDS: the bonds (if applicable) in molecules
- ANGLE: the bonds (if applicable) in molecules
- DIHEDRAL: the bonds (if applicable) in molecules
- IMPROPER: the bonds (if applicable) in molecules
- (other sections such as cross terms)

The code currently skips the title section and reads the bonds, angles, dihedrals and impropers.

A few important **Notes** / **Warnings**:

- The PSF file format is a highly redundant file format. It repeats identical topology of thousands of molecules of a common kind in some cases. GOMC follows the same approach as NAMD, allowing this excess information externally and compiling it in the code.
- Other sections (e.g. cross terms) contain unsupported or legacy parameters and are ignored.
- Following the restrictions of VMD, the order of the PSF atoms must match the order in the PDB file.
- Improper entries are read and stored, but are not currently used. Support will eventually be added for this.

The PSF file is typically generated using PSFGen. It is convenient to make a script, such as the example below, to do this:

```
psfgen << ENDMOL
topology ./Top_branched_Alaknes.inp
segment ISB{
  pdb ./STEP2_ISB_packed_BOX_0.pdb
  first none
  last none
}

coordpdb ./STEP2_ISB_packed_BOX_0.pdb ISB

writepsf ./STEP3_START_ISB_sys_BOX_0.psf
writepdb ./STEP3_START_ISB_sys_BOX_0.pdb
```

Typically, one script is run per box to generate a finalized PDB/PSF for that box. The script requires one additional file, the NAMD-style topology file. While GOMC does not directly read or interact with this

file, it's typically used to generate the PSF and, hence, is considered one of the integral file types. It will be briefly discussed in the following section.

Here's a peek at how the generated PSF file looks for a packed isobutane system (abridged):

```

PSF
  3 !NTITLE
REMARKS original generated structure x-plor psf file
REMARKS topology ./Top_Branched_Alkanes.inp
REMARKS segment ISB { first NONE; last NONE; auto angles dihedrals }

  4000 !NATOM
    1 ISB      1      ISB   C1    CH1    0.000000  13.0190  0
    2 ISB      1      ISB   C2    CH3    0.000000  15.0350  0
    3 ISB      1      ISB   C3    CH3    0.000000  15.0350  0
    4 ISB      1      ISB   C4    CH3    0.000000  15.0350  0
    5 ISB      2      ISB   C1    CH1    0.000000  13.0190  0
    6 ISB      2      ISB   C2    CH3    0.000000  15.0350  0
    7 ISB      2      ISB   C3    CH3    0.000000  15.0350  0
    8 ISB      2      ISB   C4    CH3    0.000000  15.0350  0
.
.
.
  3997 ISB      1000 ISB   C1    CH1    0.000000  13.0190  0
  3998 ISB      1000 ISB   C2    CH3    0.000000  15.0350  0
  3999 ISB      1000 ISB   C3    CH3    0.000000  15.0350  0
  4000 ISB      1000 ISB   C4    CH3    0.000000  15.0350  0
  3000 !BOND:      bonds
    1 2          1 3      1 4      5      6
    5 7          5 8
.
.
.
  3997 3998      3997 3998 3999 3997 4000
  3000 !NTHETA:  angles
    2 1          4 2      1 3      3      1      4
    6 5          8 6      5 7      7      5      8
.
.
.
  3998 3997      4000 3998 3997 3999 3999      3997 4000

    0 !NPHI: dihedrals
    0 !NIMPHI: impropers
    0 !NDON: donors
    0 !NACC: acceptors
    0 !NNB
    0 0          0 0      0 0      0      0
    0 0          0 0      0 0      0      0
.
.
.

```

### 8.3 Topology File

The topology is a whitespace separated file format, which contains a list of atoms and their corresponding masses, and a list of residue information (charges, composition, and topology). Essentially, it is a non-redundant lookup table equivalent to the PSF file.

This is followed by a series of residues, which tell PSFGen what atoms are bonded to a given atom. Each residue is comprised of four key elements:

- A header beginning with the keyword RESI with the residue name and net charge
- A body with multiple ATOM entries (not to be confused with the PDB-style entries of the same name), which list the partial charge on the particle and what kind of atom each named atom in a specific molecule/residue is.
- A section of lines starting with the word BOND contains pairs of bonded atoms (typically 3 per line)
- A closing section with instructions for PSFGen.

Here's an example of a residue definition for isobutane:

```
RESI ISB      0.00 !  isobutane - TraPPE
GROUP
ATOM C1 CH1 0.00 !  C3
ATOM C2 CH3 0.00 !  C2-C1
ATOM C3 CH3 0.00 !  C4
ATOM C4 CH3 0.00 !
BOND C1 C2 C1 C3 C1 C4
PATCHING FIRS NONE LAST NONE
```

Here's a full parameter file prepared to pack a system of isobutane:

```
*
* Custom top file -- branched alkanes
*
1 1
!
MASS 1 CH3 15.035 C !
MASS 2 CH1 13.019 C !

RESI ISB      0.00 !  isobutane - TraPPE
GROUP
ATOM C1 CH1 0.00 !  C3
ATOM C2 CH3 0.00 !  C2-C1
ATOM C3 CH3 0.00 !  C4
ATOM C4 CH3 0.00 !
BOND C1 C2 C1 C3 C1 C4
PATCHING FIRS NONE LAST NONE

END
```

Note that the keyword END must be used to terminate this file and keywords related to the auto-generation process must be placed near the top of the file, after the MASS definitions.

More in-depth information can be found in the following links:

“Topology Tutorial” (PDF, in-depth)

<http://www.ks.uiuc.edu/Training/Tutorials/science/topology/topology-tutorial.pdf>

“NAMD Tutorial: 4. Examining the Topology File”

<http://www.ks.uiuc.edu/Training/Tutorials/science/topology/topology-html/node4.html>

“Developing Topology and Parameter Files”

<http://www.ks.uiuc.edu/Training/Tutorials/science/forcefield-tutorial/forcefield-html/node6.html>

“NAMD Tutorial: 25. Topology Files”

<http://www.ks.uiuc.edu/Training/Tutorials/namd/namd-tutorial-win-html/node25.html>

**NOTE:** Links are courtesy of UIUC.

## 8.4 Parameter File(s):

Currently, GOMC uses a single parameter file and the user has the two kinds of parameter file choices:

- “CHARMM” (Chemistry at Harvard Molecular Mechanics) compatible parameter file
- “EXOTIC” parameter file

If the parameter file type is not specified or if the chosen file is missing, an error will result.

Both force field file options are whitespace separated files with sections preceded by a tag. When a known tag (representing a molecular interaction in the model) is encountered, reading of that section of the force field begins. Comments (anything after a \* or !) and whitespace are ignored. Reading concludes when the end of the file is reached or another section tag is encountered.

### CHARMM format parameter file

CHARMM contains a widely used model for describing energies in Monte Carlo and molecular dynamics simulations. It is intended to be compatible with other codes that use such a format, such as NAMD. For a general overview of the CHARMM force field, see:

[http://www.charmmtutorial.org/index.php/The\\_Energy\\_Function](http://www.charmmtutorial.org/index.php/The_Energy_Function)

Here’s the basic CHARMM contributions that are supported in GOMC:

$$\begin{aligned}U_{bond} &= \sum_{bonds} K_b(b - b^0)^2 \\U_{angle} &= \sum_{angles} K_\theta(\theta - \theta^0)^2 \\U_{dihedral} &= \sum_{dihedrals} K_\phi(1 + \cos(n\phi - \delta)) \\U_{LJ} &= \sum_{nonb,pairs} \epsilon_{ij} \left[ \left( \frac{r_{ij}^{min}}{r_{ij}} \right)^{12} - 2 \left( \frac{r_{ij}^{min}}{r_{ij}} \right)^6 \right] \\U_{LJ} &= \sum_{elec} \frac{q_i q_j}{4\pi\epsilon r_{ij}}\end{aligned}$$

As seen above, the following are recognized, read and used:

- BONDS
  - Quadratic expression describing bond stretching based on bond length (b) in Angstrom
  - Typically, it is ignored as bonds are rigid for Monte Carlo simulations. To specify that it is to be ignored, put a very large value i.e. “99999999999” for  $K_b$ .

**NOTE:** GOMC does not sample bond stretch.

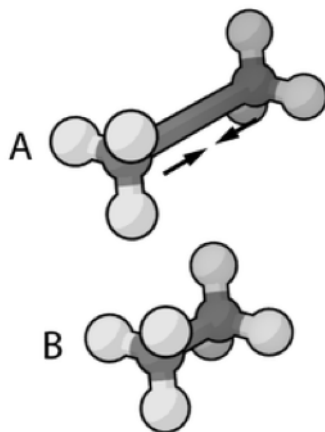


Figure 3: Image Courtesy of Wikimedia Commons

- ANGLES

- Describe the conformational behavior of an angle ( $\vartheta$ ) between three atoms, one of which is shared branch point to the other two. To fix any angle and ignore the related angle energy, put a very large value i.e. “999999999999” for  $K_\theta$ .

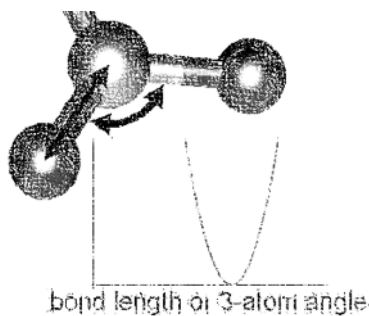


Figure 4: Image Courtesy of Wikimedia Commons

- DIHEDRALS

- Describes crankshaft-like rotation behavior about a central bond in a series of three consecutive bonds (rotation is given as  $\phi$ ).

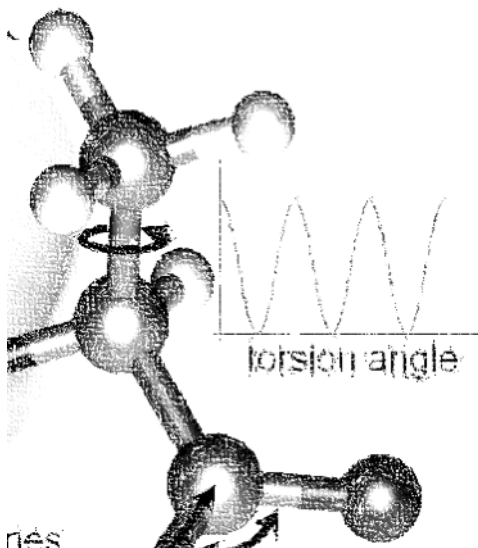


Figure 5: Image Courtesy of Wikimedia Commons

- **NONBONDED**

- This tag name only should be used if CHARMM force files are being used. This section describes 12-6 (Lennard-Jones) non-bonded interactions. Non-bonded parameters are assigned by specifying atom type name followed by polarizabilities (which will be ignored), minimum energy, and (minimum radius)/2. In order to modify 1-4 interaction, a second polarizability (again, will be ignored), minimum energy, and (minimum radius)/2 need to be defined; otherwise, the same parameter will be considered for 1-4 interaction.

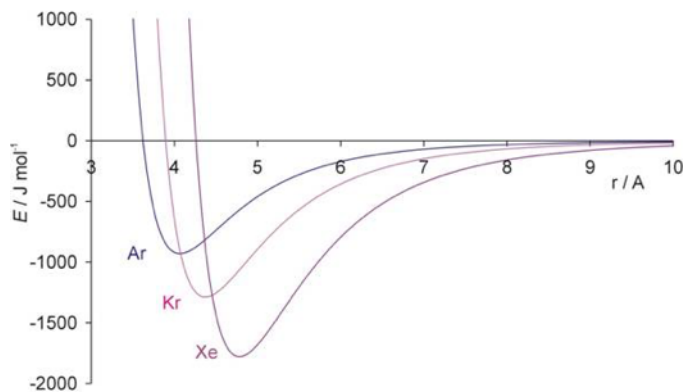


Figure 6: Image Courtesy of Wikimedia Commons

- **NBFIX**

- This tag name only should be used if CHARMM force field is being used. This section allows interaction between two pairs of atoms to be modified, done by specifying two atom type names followed by minimum energy and minimum radius. In order to modify 1-4 interaction, a second minimum energy and minimum radius need to be defined; otherwise, the same parameter will be considered for 1-4 interaction.

**NOTE:** Please pay attention that in this section we define minimum radius, not (minimum radius)/2 as it is defined in the NONBONDED section.

Currently, supported sections of the CHARMM compliant file include BONDS, ANGLES, DIHEDRALS, NONBONDED, NBFIX. Other sections such as CMAP are not currently read or supported.

### 8.4.1 BONDS

(“bond stretching”) is one key section of the CHARMM-compliant file. Units for the  $K_b$  variable in this section are in kcal/mol; the  $b_0$  section (which represents the equilibrium bond length for that kind of pair) is measured in Angstroms.

```
BONDS
!V(bond) = Kb(b - b0)**2
!
!Kb:  kcal/mole/A**2
!b0:  A
!
! Kb (kcal/mol) = Kb (K) * Boltz.  const.;
!
!atom type Kb b0 description
CH3 CH1 9999999999 1.540 ! TraPPE 2
```

**NOTE:** The  $K_b$  value may appear odd, but this is because a larger value corresponds to a more rigid bond. As Monte Carlo force fields (e.g. TraPPE) typically treat molecules as rigid constructs,  $K_b$  is set to a large value - 9999999999. Sampling bond stretch is not supported in GOMC.

### 8.4.2 ANGLES

(“bond bending”), where  $\theta$  and  $\theta_0$  are commonly measured in degrees and  $K_\theta$  is measured in kcal/mol/K. These values, in literature, are often expressed in Kelvin (K). To convert Kelvin to kcal/mol/K, multiply by the Boltzmann constant –  $K_b$ , 0.0019872041 kcal/mol. In order to fix the angle, it requires to set a large value for  $K_\theta$ . By assigning a large value like 9999999999, specified angle will be fixed and energy of that angle will be considered to be zero.

Here is an example of what is necessary for isobutane:

```
ANGLES
!
!V(angle) = Ktheta(Theta - Theta0)**2
!
!V(Urey-Bradley) = Kub(S - S0)**2
!
!Ktheta:  kcal/mole/rad**2
!Theta0:  degrees
!S0:  A
!
! Ktheta (kcal/mol) = Ktheta (K) * Boltz.  const.
!
!atom types Ktheta Theta0 Kub(?) S0(?)
CH3 CH1 CH3 62.100125 112.00 ! TraPPE 2
```

Some CHARMM ANGLES section entries include Urey-Bradley potentials ( $K_{ub}$ ,  $b_{ub}$ ), in addition to the



standard quadratic angle potential. The constants related to this potential function are currently read, but the logic has not been added to calculate this potential function. Support for this potential function will be added in later versions of the code.

The final major bonded interactions section of the CHARMM compliant parameter file are the DIHEDRALS. Each dihedral is composed of a dihedral series of 1 or more terms. Often, there are 4 to 6 terms in a dihedral. Angles for the dihedrals' deltas are given in degrees.

Since isobutane has no dihedral, here are the parameters pertaining to 2,3-dimethylbutane:

```
DIHEDRALS
!
!V(dihedral) = Kchi(1 + cos(n(chi) - delta))
!
!Kchi: kcal/mole
!n: multiplicity
!delta: degrees
!
! Kchi (kcal/mol) = Kchi (K) * Boltz.  const.
!
!atom types Kchi n delta description
X CH1 CH1 X -0.498907 0 0.0 ! TraPPE 2
X CH1 CH1 X 0.851974 1 0.0 ! TraPPE 2
X CH1 CH1 X -0.222269 2 180.0 ! TraPPE 2
X CH1 CH1 X 0.876894 3 0.0 ! TraPPE 2
```

**NOTE:** The code allows the use of 'X' to indicate ambiguous positions on the ends. This is useful because this kind is often determined solely by the two middle atoms in the middle of the dihedral, according to literature.

### 8.4.3 IMPROPERs

Energy parameters used to describe out-of-plane rocking are currently read, but unused. The section is often blank. If it becomes necessary, algorithms to calculate the improper energy will need to be added.

The next section of the CHARMM style parameter file is the NONBONDED. In order to use TraPPE this section of the CHARMM compliant file is critical. Here's an example with our isobutane potential model:

```
NONBONDED
!
!V(Lennard-Jones) = Eps,i,j[(Rmin,i,j/ri,j)**12 - 2(Rmin,i,j/ri,j)**6]
!
!atom ignored epsilon Rmin/2 ignored eps,1-4 Rmin/2,1-4
!
CH3 0.0 -0.194745992 2.10461634058 0.0 0.0 0.0 ! TraPPE 1
CH1 0.0 -0.019872040 2.62656119304 0.0 0.0 0.0! TraPPE 2
End
```

**NOTE:** The  $R_{min}$  is different from  $\sigma$ .  $\sigma$  is the distance to the x-intercept (where interaction energy goes from being repulsive to positive).  $R_{min}$  is the potential well-depth, where the attraction is maximum. To convert  $\sigma$  to  $R_{min}$ , simply multiply  $\sigma$  by 0.56123102415, and flag it with a negative sign.

The last section of the CHARMM style parameter file is the NBFIX. In this section, individual pair interaction will be modified. First, pseudo non-bonded parameters have to be defined in NONBONDED and modified in NBFIX. Here's an example if it is required to modify interaction between CH3 and CH1 atoms:

```
NBFIX
!V(Lennard-Jones) = Eps,i,j[(Rmin,i,j/ri,j)**12 - 2(Rmin,i,j/ri,j)**6]
!
!atom atom epsilon Rmin eps,1-4 Rmin,1-4
CH3 CH1 -0.294745992 1.10461634058 !
End
```