

---

# OMEGA C++ Toolkit

*version 2.3.2*

OpenEye Scientific Software, Inc.

September 10, 2008

9 Bisbee Ct, Suite D  
Santa Fe, NM 87508  
[www.eyesopen.com](http://www.eyesopen.com)  
[support@eyesopen.com](mailto:support@eyesopen.com)

Copyright © 1997-2008 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copy-right notice is precautionary only and does not imply publication or disclosure.

The information supplied in this document is believed to be true but no liability is assumed for its use or the infringement of the rights of others resulting from its use. Information in this document is subject to change without notice and does not represent a commitment on the part of OpenEye Scientific Software.

This package is sold/licensed/distributed subject to the condition that it shall not, by way of trade or otherwise, be lent, re-sold, hired out or otherwise circulated without OpenEye Scientific Software's prior consent, in any form of packaging or cover other than that in which it was produced. No part of this manual or accompanying documentation, may be reproduced, stored in a retrieval system on optical or magnetic disk, tape, CD, DVD or other medium, or transmitted in any form or by any means, electronic, mechanical, photocopying recording or otherwise for any purpose other than for the purchaser's personal use without a legal agreement or other written permission granted by OpenEye.

This product should not be used in the planning, construction, maintenance, operation or use of any nuclear facility nor the flight, navigation or communication of aircraft or ground support equipment. OpenEye Scientific software, shall not be liable, in whole or in part, for any claims arising from such use, including death, bankruptcy or outbreak of war.

Windows is a registered trademark of Microsoft Corporation. Apple and Macintosh are registered trademarks of Apple Computer, Inc. AIX and IBM are registered trademarks of International Business Machines Corporation. UNIX is a registered trademark of the Open Group. RedHat is a registered trademark of RedHat, Inc. Linux is a registered trademark of Linus Torvalds. Alpha is a trademark of Digital Equipment Corporation. SPARC is a registered trademark of SPARC International Inc.

SYBYL is a registered trademark of TRIPOS, Inc. MDL is a registered trademark and ISIS is a trademark of MDL Information Systems, Inc. SMILES, SMARTS, and SMIRKS may be trademarks of Daylight Chemical Information Systems. Macromodel is a trademark of Schrödinger, Inc. Schrödinger, Inc may be a wholly owned subsidiary of the Columbia University, New York.

Python is a trademark of the Python Software Foundation. Java is a trademark or registered trademark of Sun Microsystems, Inc. in the U.S. and other countries.

“The forefront of chemoinformatics” is a trademark of Daylight Chemical Information Systems, Inc.

Other products and software packages referenced in this document are trademarks and registered trademarks of their respective vendors or manufacturers.

# CONTENTS

|            |  |           |
|------------|--|-----------|
| <b>I</b>   | <b>Introduction</b>                    | <b>1</b>  |
| <b>1</b>   | <b>Overview</b>                        | <b>2</b>  |
| <b>2</b>   | <b>Theory</b>                          | <b>3</b>  |
| 2.1        | Filtering . . . . .                    | 4         |
| 2.2        | Stereochemistry Enumeration . . . . .  | 4         |
| 2.3        | Fragment Library Generation . . . . .  | 5         |
| <b>II</b>  | <b>Applications</b>                    | <b>6</b>  |
| <b>3</b>   | <b>Using Omega</b>                     | <b>7</b>  |
| 3.1        | Command Line Interface . . . . .       | 7         |
| 3.2        | Force Fields . . . . .                 | 13        |
| 3.3        | Torsion Library Format . . . . .       | 14        |
| 3.4        | Example Executions . . . . .           | 14        |
| <b>4</b>   | <b>Creating A Fragment Library</b>     | <b>16</b> |
| 4.1        | Theory . . . . .                       | 16        |
| 4.2        | Usage . . . . .                        | 16        |
| <b>5</b>   | <b>Flipper</b>                         | <b>19</b> |
| 5.1        | Theory . . . . .                       | 19        |
| 5.2        | Usage . . . . .                        | 20        |
| <b>6</b>   | <b>Installation and Platform Notes</b> | <b>22</b> |
| 6.1        | Licenses . . . . .                     | 22        |
| 6.2        | Installation . . . . .                 | 22        |
| 6.3        | PVM . . . . .                          | 23        |
| 6.4        | PVM hosts file . . . . .               | 23        |
| 6.5        | PVM Scaling . . . . .                  | 24        |
| <b>III</b> | <b>Toolkit</b>                         | <b>25</b> |
| <b>7</b>   | <b>OEOmega Classes</b>                 | <b>26</b> |
| 7.1        | OEOmega . . . . .                      | 26        |

|           |                                    |           |
|-----------|------------------------------------|-----------|
| <b>8</b>  | <b>OEOmega Functions</b>           | <b>34</b> |
| 8.1       | OEFlipper . . . . .                | 34        |
| 8.2       | OEOmegaGetLibraryRelease . . . . . | 34        |
| 8.3       | OEOmegaGetLibraryVersion . . . . . | 34        |
| 8.4       | OEOmegaGetPlatform . . . . .       | 34        |
| 8.5       | OEOmegaGetRelease . . . . .        | 34        |
| 8.6       | OEOmegaGetVersion . . . . .        | 34        |
| 8.7       | OEOmegaIsLicensed . . . . .        | 35        |
| 8.8       | OESliceEnsemble . . . . .          | 35        |
| <b>9</b>  | <b>Release Notes</b>               | <b>36</b> |
| 9.1       | Omega 2.3.2 . . . . .              | 36        |
| 9.2       | Omega 2.3.1 . . . . .              | 36        |
| 9.3       | Omega 2.3.0 . . . . .              | 37        |
| 9.4       | Omega 2.2.2 . . . . .              | 38        |
| 9.5       | Omega 2.2.1 . . . . .              | 38        |
| <b>IV</b> | <b>Appendices</b>                  | <b>39</b> |
| <b>A</b>  | <b>Param Files</b>                 | <b>40</b> |
|           | <b>Bibliography</b>                | <b>41</b> |

# LISTINGS

## **Part I**

# **Introduction**

# Overview

Small molecule conformer generation is a hard problem. Very small molecules like acetone and ethane are not tremendously challenging. Add only a few more atoms to the simplest organic molecules and conformer generation becomes factorially more problematic. Models used for gas-phase conformer generation are inherently inaccurate. Model quality is further degraded by attempting to incorporate something of the solution phase or protein bound conformational preferences. The latter is made especially difficult when the structure of the macromolecule to which the small molecules may bind has not been determined. Yet these are exercises routinely performed using conformer generation software. Inaccurate models and incomplete information are coupled to conformer generation engines that attempt to approach the intractably large search space of shapes that small molecules might adopt. A modicum of hubris is required to undertake such a task, and further expect it to be *useful*.

Molecular modelling results routinely depend on the quality of conformers directed into a workflow upon which results are rationalized and predictions are made. Prior to version 2.0, Omega was designed primarily to support high throughput virtual screening exercises. Reproducing bioactive conformations was formerly the focus of the program to the exclusion of all other possible exercises. A number of changes have been introduced in Omega version 2.0 that make it a more general purpose conformer generation program. Higher quality force fields (MMFF and derivatives) replace the Dreiding force field. User defined selection of the force field is possible, providing a means of biasing searches toward conformations with particular attributes. 3D model construction has been redesigned to capture non-idealized bond angle geometries while maintaining a level of symmetry appropriate for rotatable bond searches. Ring conformer generation has been made exhaustive within the limits of a force field and optimizer's ability to identify local minima. With the new features in version 2.0, Omega is a more suitable tool for low throughput conformational analysis. In addition, high throughput conformer generation and the ability to reproduce bioactive conformations has improved as a result of the additional features.

# Theory

Omega is composed of two main components; model building and torsion driving. The components are independent and can be used separately from each other. Models can be generated without performing a torsion search. Model generation may be bypassed by importing structures from external sources.

Omega builds initial models of structures by assembling fragment templates along sigma bonds. Input molecules graphs are fragmented at exocyclic sigma, and carbon to heteroatom acyclic (but not exocyclic) sigma bonds. Conformations for the fragments are either retrieved from pregenerated libraries built with `makefraglib`, or constructed on-the-fly using the same distance constraints followed by geometry optimization protocol that `makefraglib` uses. Molecule assembly is accomplished by simple vector alignment since all inter-fragment joints are along sigma bonds.

Once an initial model of a structure is constructed, or given as input, Omega generates additional models by enumerating ring conformations and invertible nitrogen atoms. Ring conformations are taken from the same fragment library used to build an initial model. Omega detaches all exocyclic substituents from a ring system, aligns and attaches them relative to the new ring conformation. Omega attempts to generate every possible combination of ring conformations possible for a given structure.

The next step in model generation is to detect and enumerate invertible nitrogens. Nitrogens that have pyramidal geometry, no stereochemistry specified, no more than one hydrogen, are three valent, and have no more than three ring bonds are considered by Omega to be invertible. Invertible in this context simply means that at room temperature a pyramidal nitrogen is likely to be able to rapidly (on an NMR timescale) interconvert between two puckered forms. All multiconformer ring models are further expanded by enumerating all possible nitrogen puckers. The resulting model set is the starting point for conformer search by torsion driving.

Omega begins the torsion search process by examining the molecular graph and determining the bonds may freely rotate. By default, Omega selects acyclic sigma bonds that have at least one non-hydrogen atom attached to each end of the bond. Hydrogen rotors (i.e. hydroxyl groups) are not altered during the torsion search. Doing so would make a combinatorially nasty problem even worse with no affect on the final ensemble. The final ensemble selection is based on heavy atom RMS distance which is unaffected by hydrogen positions. A list of possible dihedral angles are then assigned to each rotatable bond. The current mechanism for assignment is based on SMARTS matching, although alternate strategies for assigning angles based on experimental (i.e. X-ray) or theoretical (i.e. fragment optimization studies) are possible. The molecular graph is then subjected to pattern and geometric symmetry detection. Common patterns such as para-disubstituted benzene are used to reduce the number of symmetry equivalent dihedral angles that need to be searched. All torsions are altered by 120 and 180 degrees, and an RMS calculation is performed taking into account symmetry equivalent atoms in order to detect two and three fold symmetries. Torsions are then grouped into fragments of sets of up to five contiguous rotatable bonds. Exhaustive depth first torsion search is performed on each of the fragments, and the resulting conformers are placed into list sorted by energy. Entire structures are assembled by combining the lowest energy set of fragments, and then the next lowest set, until the search is terminated. Termination conditions include a limit on the total number of conformers that may be generated, fragment list may be exhausted, or the sum of the fragment energies

exceeds the energy window of the global minimum structure. The best conformers identified in the torsion search are rank ordered based on energy. A final ensemble is selected by sequentially testing the conformers using the RMS distance cutoff. To be accepted in to the final ensemble, a conformer must have an RMS distance to every other member of the ensemble that exceeds the user defined cutoff value. The final ensemble is populated up to the user defined maximum ensemble size limit, or until the list of low energy conformers is exhausted.

Input File Preparation

## 2.1 Filtering

Filtering based on graph (and possibly physical) properties should always be carried out prior to generating multi-conformer databases using Omega. Eliminating undesirable compounds prior to generating conformers will save execution time of both Omega and down stream applications, and space on disk. Large polypeptides or proteins, very flexible molecules, or simply molecules that would never be considered useful for the ultimate modeling application are best eliminated from a data set as early as possible.

The most important graph filter to apply is rotatable bond count. Although Omega may be able to generate conformers for molecules with more than 20 rotatable bonds, the results of such an exercise would be dubious at best for any conformer generation method. Number of rings, especially flexible rings, should also be used to exclude irrelevant molecules. Omega will handle molecules that have many thousands of possible ring combinations, although the time expenditure may be prohibitive and the results equally questionable to molecules with an unreasonably large number of rotatable bonds. Simple element filters may be useful as well, although Omega will discard compounds for which no force field parameters exist. Using element filters beforehand may simply aid in tracking the rationale for discarding compounds instead of searching through Omega log files for failure modes.

The `filter` program from OpenEye Scientific Software provides all of the functionality outlined above, and additional physical property filters. It is highly recommended that `filter` or a program similar in functionality be used for input file preparation of large datasets.

## 2.2 Stereochemistry Enumeration

Compounds that contain unspecified or ambiguous definitions of stereochemistry may be preprocessed before generating conformers to add explicitly specify stereochemistry. Input molecules that have three dimensional coordinates inherently have stereochemistry specified, but SMILES or two dimensional SD files may have atoms (R/S) or bonds (E/Z) for which the stereochemistry is unknown or unspecified. Omega will generate structures for compounds of unknown configuration, however, in virtual screening exercises it may be beneficial to simply enumerate possible stereoisomers and treat each stereoisomer as a separate compound.

Omega distributions contain a utility called `flipper` that enumerates unspecified stereochemistry within user defined limits. For an explanation of how to use `flipper`, consult Section 5. Stereochemistry enumeration is an exponential task. For every atom or bond (N) in a molecule that has two possible stereochemical 'states', there are  $2^N$  possible stereoisomers for the molecule. Enumerating all possible stereoisomers for molecules may be unreasonable in terms of CPU time or storage space. `flipper` has user defined limits to regulate the enumeration process and keep it practical for routine applications.

Enumerating stereoisomers provides an information gain that may aid in operations downstream from conformer generation. `flipper`, or a similar stereochemistry enumeration tool should be used prior to generating conform-

ers with Omega.

## 2.3 Fragment Library Generation

Prior to conformer generation, Omega builds a set of three dimensional models of a compound that contain the bond lengths, angles, and ring conformations that will be held fixed during the torsion search. Omega is capable of generating fragment templates on-the-fly, however, using pregenerated templates is far more efficient and will speed execution of conformer generation. Omega distributions include a program called `makefraglib` that can be used to create libraries of molecular fragments and ring conformers that can be used by Omega to build three dimensional models of molecules. For an explanation of how to use `makefraglib`, consult Section 4. In practice, extensive fragment libraries need only to be constructed a single time and rarely need to be altered. Corporate and vendor databases can be used to construct an extensive fragment library. Once built, the fragment libraries will rarely need to be updated, and will provide a significant enhancement in performance.

**Part II**

**Applications**

# Using Omega

## 3.1 Command Line Interface

Executing Omega with no arguments will result in:

```
prompt> omega2

No argument specified on the command line
Required parameters:
  -in : Input filename
  -out : Output filename
For more help type:
  omega2 --help'
```

A description of the command line interface can be obtained by executing omega with the `-help` option.

```
prompt> omega2 --help
```

will generate the following output:

```
Help functions:
omega2 --help simple      : Get a list of simple parameters
omega2 --help all         : Get a complete list of parameters
omega2 --help defaults    : List the defaults for all parameters
omega2 --help <parameter> : Get detailed help on a parameter
omega2 --help html        : Create an html help file for this program
```

If you desire to see all of the command-line options use `-help all`.

```
prompt> omega2 --help all
```

will generate the following output:

```
Complete parameter list
File Options
  -commentEnergy : Writes conformer energies to the comment field
  -in : Input filename
  -includeInput : Pass the input structure into output (unmodified)
  -log : Override prefix used to name output files
  -out : Output filename
  -param : Omega control parameter file
  -pendingFile : Filename used for pending molecules file
  -prefix : Prefix to use to name output files
  -rotorOffsetCompress : Output compressed OEBinary
  -sdEnergy : Writes conformer energies to the SD tag field
  -status : Filename used for status report file
  -verbose : Triggers copious logging output
  -warts : Add conformer number wart to title of multiconformers

3D Construction Parameters
  -addfraglib : File(s) containing fragments used for 3D construction, in
              addition to built-in fraglib.
  -buildff : Force field to use for initial structure refinement
  -canonOrder : Place atoms/bond in canonical order
  -deleteFixHydrogens : Deletes hydrogens from fixfile fragment
  -dielectric : Dielectric used for electrostatic calculations
  -exponent : Exponent used for electrostatic calculations
  -fixfile : File used to specify a user defined fragment
  -fixrms : RMS used to identify invalid superpositions onto fixfile
           template
  -fromCT : Generate structures from connection-table only.
  -maxmatch : Maximum number of allowed matches of fixfile fragment
  -setfraglib : File(s) containing fragments used for 3D construction.
              Replaces built-in fraglib
  -umatch : Only use unique matches for fixfile fragment replacement

Structure Enumeration
  -enumNitrogen : Enumerate pyramidal invertible nitrogen geometries
  -enumRing : Enumerate ring conformers

Torsion Driving Parameters
  -erange : Energy window values based on number of rotors
  -ewindow : Energy window used for conformer selection
  -maxConfRange : Maximum number of conformers based on number of rotors
  -maxconfgen : Maximum number of conformations to be generated
  -maxconfs : Maximum number of conformations to be saved
  -maxrot : Maximum number of rotatable bonds in a molecule
  -maxtime : Maximum time (s) allowed for torsion search
  -rangeIncrement : Size of rotor groupings for range flags
  -rms : RMS threshold used to determine duplicate conformations
  -rmsrange : RMS threshold values based on number of rotors
  -searchff : Force field to use during torsion search
  -torlib : Input name of torsion rules file

PVM Parameters
  -pvmconf : A text file specifying a PVM configuration
  -pvmdebug : Generate an enormous volume of PVM debug information
  -pvmlog : Filename used for PVM log file.
  -pvmpass : Number of molecules to pass to a slave at one time
```

### 3.1.1 Required Parameters

- in** File containing one or more molecular connection tables to be processed by Omega. File format types are discussed in section 3.1.2.
- out** File to write conformers generated by Omega. File formats are discussed in section 3.1.2. Gzipped OEBinary is the recommended output format.

### 3.1.2 Molecular File Formats

Omega can read and write a variety of molecular file formats. The file format is automatically interpreted from the filename suffix.

| File type    | Extension                              |
|--------------|--|
| SMILES       | .smi .ism .can .smi.gz .ism.gz .can.gz |
| SDF          | .sdf .mol .sdf.gz .mol.gz              |
| SKC          | .skc .skc.gz                           |
| CDK          | .cdk .cdk.gz                           |
| MOL2         | .mol2 .mol2.gz                         |
| PDB          | .pdb .ent .pdb.gz .ent.gz              |
| MacroModel   | .mmod .mmod.gz                         |
| OEBinary v2  | .oeb <b>.oeb.gz</b>                    |
| Old OEBinary | .bin                                   |

Old OEBinary format can be read but not written by Omega. Gzipped OEBinary version 2 (.oeb.gz) is the recommended output format.

Omega is also capable of piping formatted input and output. The simple "-" can be used in place of a file name to indicate `std::cin` or `std::cout` with the default SMILES format.

```
prompt> omega -in - -out -
```

This execution will run Omega with `std::cin` as the input with SMILES format. It will also open `std::cout` with SMILES format as output. However, the use of "-" does not allow control of the file format.

To control the format of `std::cin` and `std::cout` one may use the file extensions without a preceding filename.

```
prompt> omega -in .ism -out .oeb.gz
```

This executes Omega with the input from `std::cin` formatted in isomeric SMILES and the output sent to `std::cout` in gzipped OEBinary version 2 format.

### 3.1.3 Optional Input/Output Parameters

- fraglib** See description in Section 3.1.4. No longer required as of Omega 2.2. (Alias `-setfraglib`)
- addfraglib** See description in Section 3.1.4.
- commentEnergy** This flag causes the conformer energy to be written in the comment field for each conformer. This is particularly useful when writing SD or MOL2 files that will be passed to software that anticipates strain energies written in the comment field. [default = false]

- param** The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supercede options found in the parameter file. Omega generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by Omega is created by combining the prefix base name (see `-prefix`) with the `.parm` extension.
- prefix** The argument for this flag defines the prefix to be used for various information and data files generated by Omega. Most important among these is the `'omega2.parm'` file which includes a copy of all the parameters used in the Omega run. The prefix is also used to generate a default log file name if not explicitly specified with the `-log` (see `-log`) flag. [default = `omega2`].
- rotorOffsetCompress** This flag controls the behavior of writing the rotor offset compressed flavor of the OEBinary format. Rotor offset compressed files contain the same information as standard cartesian OEBinary files, but require a small fraction of the storage space. Optimal compression of Omega output can be obtained by writing GZip compressed OEBinary files with the `-rotorOffsetCompress` flag turned on. [default = `true`]
- sdEnergy** The `-sdEnergy` flag controls the behavior of writing strain energies as SD tags. This flag may be used with either OEBinary or MDL SD files. [default = `false`]
- log** The argument for this flag specifies the name of the log file. The level of detail for logfile information can be altered using the `-verbose` (see `-verbose`) flag. Output can be directed to the terminal instead of a file by giving a hyphen `'-'` as the argument to the flag instead of a filename. Generation of an output log may be disabled by providing `'nul'` or `'null'` as an argument. [default = `prefix.log`]
- verbose** This is a boolean flag that controls the level of detail written to the log file. By default Omega will only write minimal information to the log file. Molecule titles and warning messages constitute the bulk of logging at the default level. Verbose logging will cause more information to be written to the log file in order to follow behavior during program execution. [default = `false`]
- warts** This boolean flag is used to generate unique titles for conformers that reflect their position in an ensemble of conformations produced by Omega. The title given to each conformer will begin with the molecule title taken from the input file, and appended with an underscore and the integer corresponding to the rank order number of the conformer in the final ensemble. [default = `false`]
- includeInput** When true this boolean flag will include the input conformer in the output file. This requires that the input file format is a 3D format (eg: SDF, MOL2, OEB, etc) [default = `false`]

### 3.1.4 3D Construction Parameters

- buildff** This flag sets the force field used for constructing fragments that are assembled to build an initial model of the input structure. The choice of force field will not affect geometries of structures taken from a file of pre-built fragments (see `-fraglib`), except in the positioning of protons attached to ring systems. Consult the description of Force Fields (Section 3.2) for an explanation of appropriate arguments for this flag. [default = `mmff94s_NoEstat`]
- canonOrder** This flag can be used to disable the automatic reordering of input molecules to a canonical atom and bond order. In order obtain consistent results from different file formats and different connection table orders, Omega reorders the input connection table. This behavior can be turned off using the `-canonOrder` flag, however, the resultant ensembles will likely be inconsistent in their composition. [default = `true`]

- deleteFixHydrogens** When a fixfile fragment is specified, all possible mappings (matches) of the fragment and the input molecule are considered for positioning the input molecule. This process begins with a substructure search of the fragment in the input molecule. If hydrogens are included on the fixfile fragment, the number of possible matches will grow exponentially with the number of equivalences. Each geminal hydrogen pair on the fixfile produces a non-productive match multiplied by all other non-productive matches. This flag can be used to prevent the explosion of non-productive matches by deleting hydrogens from the fixfile fragment prior to the substructure search. [default = false]
- dielectric** The argument to this flag allows the user to specify the dielectric applied to the coulomb term of the force field. This flag may only be used with a version of the search force field that includes the coulomb term, such as the MMFF94 and MMFF94s variants. [default = 1.0]
- exponent** The argument to this flag allows the user to specify the exponent applied to the inverse distance calculation of the coulomb term of the force field (i.e.  $(1/r)^X$  where X is the argument to the `-exponent` flag). This flag may only be used with a version of the search force field that includes the coulomb term, such as the MMFF94 and MMFF94s variants. [default = 1.0]
- fixfile** The argument that follows this flag is a molecule file used to specify the coordinates for a substructure of the input molecules. An initial structure is generated for every input molecule, and then a substructure search is performed using the molecule or fragment provided in the fixfile as a query molecule. Every instance of the fixed substructure found in the input molecule up to a predetermined limit (see `maxmatch` and `umatch`) is used to replace the coordinates of the atoms that match the substructure. The input molecule coordinates are aligned relative to the substructure prior to fragment replacement, and then the coordinates are taken from the fixed fragment and assigned to the corresponding atoms of the input molecule. A separate alignment, replacement, and then conformer search is carried out for every matching substructure in the input molecule.
- fixrms** Fixfile fragments taken from crystallographic sources may differ in their geometry relative to optimal MMFF geometries. Omega attempts to superimpose built structures onto fixfile fragments, and if the geometry differs too greatly Omega considers the superposition a poor match and will fail to build a structure using the fixfile. This flag can be used to loosen the default RMS superposition criteria to allow suboptimal superpositions to succeed in spite of the poor geometric complementarity. [default = 0.15]
- setfraglib** The argument that follows this flag is an OEBinary molecule file containing the coordinates of pre-built acyclic fragments, and multiple conformations of cyclic systems. Starting with Omega 2.2, this flag is no longer required, as a default fragment library has been built into the program. Although Omega can generate fragment 3D geometries on the fly, building them in advance speeds execution significantly. Multiple fragment files may be provided as a comma delimited list of filenames. The fragment file(s) chosen should be constructed using a force field corresponding to the one being used for the torsion search. For example, the MMFF94s variant of the Merck Molecular Force Field should be used both for fragment construction and torsion search. Normally this file is build using the `makefraglib` auxilliary program.
- addfraglib** The argument that follows this flag is one or more fragment files, normally generated by `makefraglib`. These fragments will supplement the built-in fragment library (new in Omega 2.2).
- fromCT** This boolean flag determines whether Omega should generate an initial set of 3D coordinates using only the connection table of the input molecule. Initial model generation is always necessary for molecule file formats devoid of coordinates (i.e SMILES). Bond lengths and angles taken from molecule files containing coordinates may be retained by setting this flag to false. [default = true]
- maxmatch** The `-maxmatch` flag is used to limit the number of fixfile substructure matches in the input molecule. Each match will result in replacement of the matching substructure with coordinates taken from the fixfile fragment. The number of matches may need need to be limited using this parameter for a substructure where many matches are possible. [default = 10]

**-umatch** The `-umatch` boolean flag determines whether only the unique substructure matches of the fixfile are used for coordinate replacement. A unique substructure match is defined as a match that does not cover the identical set of target atoms as any other substructure match in a set. For example, a benzene substructure will match a benzene ring 12 times. Only one substructure match constitutes a unique match, while the other 11 matches are duplicates. If the flag is set to false then all possible substructure matches may be used for coordinate replacement. This behavior is usually unnecessary as non-unique matches will frequently lead to duplication. [default = true]

### 3.1.5 Structure Enumeration

**-enumNitrogen** The `-enumNitrogen` boolean flag controls the behavior of Omega with respect to enumeration of non-planar nitrogens. Any nitrogen with pyramidal geometry in the initial model of the input molecule, and having no more than two ring bonds is considered by Omega to be 'invertible'. Omega will enumerate all possible puckers of all invertible nitrogens if the `-enumNitrogens` flag is set to true. [default = true]

**-enumRing** The `-enumRing` boolean flag controls the behavior of Omega with respect to ring conformations. If this flag is set to true, Omega will generate all possible combinations of all ring conformations in a molecule. Ring systems with only a single conformation will be replaced with a conformation taken from a fragment file, or generated on the fly by Omega. If this flag is set to false then no ring conformer enumeration or replacement will occur. Initial geometries provided in by an input file (see `-fromCT false`) may therefore be preserved by setting `-enumRing` to false as well. [default = true]

### 3.1.6 Torsion Driving Parameters

**-erange** The `-erange` flag sets the energy cutoff used as an accept or reject criteria for conformers depending on the number of rotatable bonds in the structure. Any conformer that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum conformer will be accepted. Conformers with strain energies above this threshold are rejected. The energy range is given as a comma separated list of values that correspond to the `-rangeIncrement` parameter. For example, `-erange '5.0, 10.0, 15.0, 20.0'` used with `-rangeIncrement 3` sets the energy window to 5.0 Kcal/mol for structures with zero to two rotatable bonds, 10.0 Kcal/mol for structures with three to five rotatable bonds, and so on. The energy window for structures with more rotors than the highest `-erange` value specified will taken as the highest specified value.

**-ewindow** The `-ewindow` flag sets the energy window used as an accept or reject criteria for conformers. Any conformer that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum conformer will be accepted. Conformers with strain energies above this threshold are rejected. [default = 10.0]

**-maxconfs** The `-maxconfs` flag sets the maximum number of conformations to be generated. Conformers are assembled in energy sorted order of the constituent fragments. The default value in most cases will vastly exceed the number of conformers that need to be generated in order to select the best possible ensemble based on the RMS distance and energy criteria. As a special case, setting `"-maxconfs 0"` will result in Omega skipping the duplicate removal step and it will write all generated conformers to the output file. Note that this implies `"-rms 0"` is also used. [default = 400]

**-maxConfRange** This string argument to this flag allows the user to specify the maximum number of conformers to be output for a structure based on the number of rotatable bonds in the structure. For example,

`-maxConfRange` `''100,200''` used with `-rangeIncrement` 5 will cause Omega to output 100 conformers for structures with zero to 4 rotors, and 200 conformers for all structures with more than 4 rotors.

**-maxconfgen** The `-maxconfgen` flag controls the number of fully constructed conformers that Omega will attempt to build. [default = 50000]

**-maxrot** The `-maxrot` flag sets the maximum number of rotatable bonds cutoff. Molecules that have equal to or fewer rotors than the `-maxrot` cutoff will be processed by Omega. Omega will not search for conformers of molecules that have more rotors than the `-maxrot` cutoff. By default, Omega does not apply a number of rotatable bond cutoff. Instead, a desired cutoff must be supplied by the user. [default = -1]

**-maxtime** This flag limits the amount of time (in seconds) spent generating conformers for each molecule. [default = 120.0]

**-rangeIncrement** The `-rangeIncrement` is used to control the number of rotatable bonds range used with the `-maxConfRange`, `-rmsrange`, and `-erange` flags. The preceding flags are used to control the maximum number of conformers, RMS cutoff, and energy windows used that are dependent on the number of rotors in a given structure. [default = 5]

**-rms** The `-rms` flag sets the minimum Root Mean Square (RMS) cartesian distance below which two conformers are duplicates. The RMS calculation is performed after superposition such that the true minimum distance between conformers is calculated. Lowering the `-rms` value may cause Omega to generate ensembles that contain more representative conformers of a similar shape. Higher `-rms` values may result in smaller, yet possibly more shape diverse ensembles. [default = 0.8]

**-rmsrange** This string argument to this flag allows the user to specify the RMS cutoff used for duplicate conformer removal based on the number of rotatable bonds in the structure. For example, `-rmsrange` `''0.8,1.0''` used with `-rangeIncrement` 5 will cause Omega to use an RMS cutoff value of 0.8 for structures with zero to 4 rotors, and an RMS cutoff value of 1.0 for all structures with more than 4 rotatable bonds.

**-searchff** This flag sets the force field used to calculate strain energies of conformers generated during a torsion search. Consult the description of Force Fields (Section 3.2) for an explanation of appropriate arguments for this flag. [default = mmff94s\_NoEstat]

**-torlib** The `-torlib` flag is used to specify the file name of the file containing rules used in resolution control of the torsion driving part of conformer generation. Refer to the section describing the format of the torsion library (3.3). If no torsion library is provided then Omega will use an internally stored copy of the default torsion library.

## 3.2 Force Fields

Omega provides the facility for users to select one of a number of pre-defined force fields. The force field selected may be different for model construction and torsion search. The ability to select a force field provides a mechanism for task specificity. Some force fields may be more appropriate for solution phase ensemble generation, whilst others may excel for bioactive conformer reproduction. The following pre-defined force fields may be used as arguments to the `-buildff` and `searchff` flags.

**mmff** Exact reproduction of the published Merck Molecular Force Field (MMFF)[14, 15, 16, 17, 18, 19, 20].

**mmff\_NoEstat** This force field variant includes all MMFF terms except coulomb interactions.

**mmff\_Trunc** This force field variant excludes both coulomb interactions and the attractive part of Van der Waals interactions. All other components of the MMFF force field are calculated according to literature specifications.

**mmff94s** Exact reproduction of the 94s variant of the Merck Molecular force Field (MMFF94s)[14, 15, 16, 17, 18, 19, 20].

**mmff94s\_NoEstat** This force field variant includes all MMFF94s terms except coulomb interactions.

**mmff94s\_Trunc** This force field variant excludes both coulomb interactions and the attractive part of Van der Waals interactions. All other components of the MMFF94s force field are calculated according to literature specifications.

### 3.3 Torsion Library Format

A file of alternate torsion rules may be specified with the `-torlib` command. Omega will match only the first rule found for a torsion angle, and discontinue matching alternate possible rules. Thus, a correctly ordered torsion file will be arranged with the most specific patterns appearing at the top of the file, and more general patterns appearing toward the end. Simple torsion rules are composed of a single SMARTS pattern containing at least four atom expressions, followed by a listing of the torsion angles that Omega will sample. Each reference atom in the SMARTS pattern that is used to define the torsion angle being sampled must have a map index (numbered 1 through 4) specified that indicates the ordering of the atoms in the torsion angle. The pattern must appear all on a single line, with a carriage return separating one rule from the next. Comments in the file must be preceded with a '#' character. The following is an example of a simple torsion rule.

```
#methyl ester pattern
[O:1]=[C:2]-[O:3][CH3:4] 0
```

More advanced rules may be included that alter the energy calculation for particular torsion angles. In these types of torsion rules, a SMARTS pattern with associated map indices is still used to define the molecular environment in which the rule is to be applied, but the sampled values appear on subsequent lines with one torsion angle per line. The first number per line indicates the torsion angle, in degrees, that Omega must sample. If a second number follows a torsion angle on the same line, the value is added to the total energy computed for that conformer. The following is an example of an advanced torsion rule.

```
#experimental structure test
O=[C:1][NX3H:2][c:3]([cH,nH0])[nH:4]
0
180 10.0
<end>
```

### 3.4 Example Executions

This section has a series of example Omega command-line executions. Each example is followed by a brief description of its behavior.

```
prompt> omega2 drugs.smi drugs.oeb.gz
prompt> omega2 -in drugs.smi -out drugs.oeb.gz
```

These two commands will yield identical results. These execute Omega with the default parameters. The file `drugs.smi` is opened in SMILES format for input, and the output is written to the file `drugs.oeb.gz` in gzipped OEBinary format.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -param myparameters
```

This command is the same as the one above except for the `-param` flag. It executes Omega with the parameters found in the `myparameters` file. The file `drugs.smi` is opened in SMILES format for input, and the output is written to the file `drugs.oeb.gz` in gzipped OEBinary format.

```
prompt> omega2 -param myparameters drugs.smi drugs.oeb.gz
prompt> omega2 drugs.smi drugs.oeb.gz -param myparameters
```

The first of these two commands will yield exactly the same results as the example above. `drugs.smi` will be mapped to the `-in` flag and `drugs.oeb.gz` will be mapped to the `-out` flag begin the second to last and last command-line arguments respectively. Unfortunately, the second of these two commands, will fail to parse because the implicit input and output arguments are not the final two arguments in the list.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -param myparameters -maxconfs
600
```

Again, this is a very similar command. It executes Omega using the parameters in the `myparameters` file, except the `-maxconfs` parameter is over-ridden with the 600 from the command line. The command-line `-maxconfs` parameter would take precedence over the value in the parameter file independent of the order of flags on the command line.

```
prompt> DBQuery "barbiturate" | omega2 -in .sdf -out .oeb.gz | vida2 .oeb.gz
```

This execution assumes that a process called "DBQuery" can be called with the parameter "barbiturate" and return a stream of molecules in MDL's `.sdf` format. This output is piped into Omega, which interprets the format correctly and generates multiconformer molecules using the default parameters. Omega writes the output to `std::cout` in gzipped OEBinary format, which is read by OpenEye's Vida2 molecular viewer.

```
prompt> omega2 -in drugs.mol2 -out drugs.oeb.gz -fromCT true
```

The `-fromCT true` flag will cause Omega to ignore the input conformations in the `drugs.mol2` file. An initial conformation will be generated by a distance-bounds algorithm from the connection-table of the molecules in the input file. Be aware that the default value of `-fromCT` is true, however it is listed explicitly here for emphasis.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -log null
```

The `-log` flag normally allows specification of the log file's name. However, `nul` and `null` are reserved names which indicate that no log file will be written.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz -log null
```

When the `-log` parameter is set to `nul` or `null`, no logging will be written (this includes failure modes).

---

# Creating A Fragment Library

`makefraglib` identifies, extracts, and generates conformers of molecular fragments. The model builder in Omega fragments input molecules, retrieves corresponding fragment conformers, and assembles them into a three dimensional structure. Pregenerating fragment conformers using `makefraglib` accelerates the model building procedure.

## 4.1 Theory

The model builder in Omega attempts to capture all of the relevant bond length and angles, and ring conformations for an input molecule. Fragment assembly allows for a 'divide and conquer' approach to model building. Much of the relevant information for building a molecular models is contained in carefully chosen fragments. `makefraglib` uses distance constraints and geometry optimization to generate fragment conformations. Only a single conformation is stored for acyclic fragments, while all possible unique ring conformers are retained within user defined limits. Libraries created with `makefraglib` can then be provided to Omega using the `-setfraglib` or `-addfraglib` flag.

## 4.2 Usage

Makefraglib can be used from the command-line to generate ring templates for use with omega starting from only a collection of molecules. The output file from `makefraglib` can then be passed directly into Omega using the `-setfraglib` or `-addfraglib` flag. Although fragment libraries may be concatenated using the Unix 'cat' command, this is not strictly necessary as multiple fragment libraries may be specified as one argument to the flag.

### 4.2.1 Command-Line Interface

A listing of all command line options for `makefraglib` along with a brief description can be obtained by executing `makefraglib` with the command line option `-help all`.

```
prompt> makefraglib --help all

Complete parameter list
  -pvmconf : A text file specifying a PVM configuration

File Options
  -in : Input filename
  -log : Override prefix used to name output files
  -out : Output filename
  -param : makefraglib control parameter file
  -prefix : Prefix to use to name output files.
  -skip : Existing fragment library to avoid duplicating
  -verbose : Verbose output

3D Construction Parameters
  -buildff : Force field to use for initial structure refinement
  -ewindow : Energy window covered by final conformers
  -fromCT : Generate structures from connection-table only.
  -rms : Minimum RMS between final conformers
  -startfact : Factor for determining number of random starts

PVM
  -pvmdebug : Generate an enormous volume of PVM debug information
  -pvmpass : Number of molecules to pass to a slave at one time
```

## 4.2.2 Parameters

- buildff** This flag sets the force field used for constructing fragments that are assembled to build an initial model of the input structure. Consult the description of Force Fields (Section 3.2) for an explanation of appropriate arguments for this flag. [default = mmff94s\_NoEstat]
- ewindow** The `-ewindow` flag sets the energy window used as an accept or reject criteria for conformers of ring systems. Any conformer that has a calculated strain energy less than the sum of the energy window and the energy of the global minimum conformer will be accepted. Conformers with strain energies above this threshold are rejected. [default = 10.0]
- in** File containing one or more molecular connection tables to be processed by Omega. File format types are discussed in section 3.1.2.
- log** This argument to this flag specifies the name of the log file. The level of detail for logfile information can be altered using the `-verbose` (see `-verbose`) flag. Output can be directed to the terminal instead of a file by giving a hyphen '-' as the argument to the flag instead of a filename. Generation of an output log may be disabled by providing 'nul' or 'null' as an argument. [default = prefix.log]
- out** File to write conformers generated by Omega. File formats are discussed in section 3.1.2. OEBinary is the recommended output format.
- param** The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supercede options found in the parameter file. `makefraglib` generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by `makefraglib` is created by combining the prefix base name (see `-prefix`) with the '.parm' extension.

- prefix** The argument for this flag defines the prefix to be used for various information and data files generated by `makefraglib`. Most important among these is the 'makefraglib.parm' file which includes a copy of all the parameters used in the Omega run. The prefix is also used to generate a default log file name if not explicitly specified with the `-log` (see `-log`) flag. [default = makefraglib].
- skip** The argument that follows the `skip` flag is a file created previously with `makefraglib`. This enables `makefraglib` to avoid re-creating fragments for which conformers have already been created. Three dimensional structures will only be generated for unique fragments found in the input molecule file.
- startfact** The floating point argument to the `-startfact` flag indirectly controls the level of sampling attempted to determine fragment conformation. The equation used to determine the number of random coordinate starting points is  $\text{number of samples} = (10 + (\sum(\text{nrb}==3) - \sum(\text{nrb}==4)) * \text{startfact})$ .  $\sum(\text{nrb}==3)$  is the sum of the number of atoms in the molecule that have exactly three ring bonds.  $\sum(\text{nrb}==4)$  is the sum of the number of atoms in the molecule that have exactly 4 ring bonds. In general, the default level of sampling will far exceed the level necessary to reproduce all reasonably low energy conformations of a ring system. [default = 20.0]
- verbose** This is a boolean flag that controls the level of detail written to the log file. By default Omega will only write minimal information to the log file. Molecule titles and warning messages constitute the bulk of logging at the default level. Verbose logging will cause more information to be written to the log file in order to follow behavior during program execution. [default = false]

### 4.2.3 Example executions

The following section shows several common command-line executions of `makefraglib`. Each example is followed by an explanation of what the program will do.

```
prompt> makefraglib -in drugs.smi -out fraglib.oeb
```

All of the molecules in `drugs.smi` will be processed. The set of all unique fragments (according to Omega fragmentation rules) from all of the molecules in `drugs.smi` will be collected.

```
prompt> makefraglib -in vendor.smi -out new_fraglib.oeb -skip  
fraglib.oeb -ewindow 5.0
```

Conformers will be generated for all of the unique fragments found in the file `vendor.smi` that are not present in `fraglib.oeb`. An energy window of 5 Kcal above the global minimum conformer will be used to accept or reject conformations of flexible ring systems.

# Flipper

## 5.1 Theory

The Omega distribution includes `flipper`, a utility program for enumerating stereocenters in a molecule. We have taken the philosophy that conformer generation and stereochemistry enumeration are related but distinct problems. Thus we provide `flipper` as part of the omega product, but not as part of the same executable.

The current version of `flipper` can handle enumeration of R/S stereochemistry (including N and S stereochemistry) as well as cis/trans stereochemistry. Flipper uses graph algorithms to determine which atoms are stereocenters. If a stereocenter has a designated stereochemistry, by default `flipper` does not change the stereochemistry. However, if a stereocenter does not have a specified stereochemistry, `flipper` will enumerate both stereochemistry states of the stereocenter.

Many common file formats are limited in their ability to fully express stereochemistry without resorting to Cartesian coordinates. This can lead to ambiguity in whether a particular conformer is supposed to represent a racemic mixture or an isolated stereoisomer. SD file format is better than most formats. It allows designation of R, S, or racemic on each tetrahedral stereocenter independent of the particular conformation. However, SD format is not as robust for cis-trans stereo. Isomeric SMILES is the only common file format which allows complete specification of R/S and cis/trans stereochemistry. In fact, though `flipper` does not currently support higher orders of stereochemistry, the Isomeric SMILES format is capable of handling them. For this reason, `flipper` output is limited to SD format (`.sdf` or `.mol`) and isomeric SMILES format (`.ism`). OpenEye OEBinary format, like isomeric SMILES, is capable of fully specifying stereochemistry, however, isomeric SMILES is more efficient for compounds without any data beyond the title and connection table.

### 5.1.1 Input dimensionality

Flipper is designed for use prior to 3D coordinate generation. So 2D or 0D (SMILES) input is normally expected. Input files containing 3D coordinates are handled, but the 3D coordinates are ignored and not transferred to the output.

## 5.2 Usage

Flipper is a simple command-line utility program with only four parameters.

### 5.2.1 Command Line Interface

If you execute flipper with no arguments, it will produce the following banner.

```
No argument specified on the command line
Required parameters:
  -in : Input filename
  -out : Output filename (.ism or .sdf format only)
For more help type:
  flipper --help'
```

As described in the output above, if you desire additional command-line help, using the following prompt will result in:

```
prompt> flipper --help all
```

```
Complete parameter list
Flipper
  -forceflip : Generate structures from connection-table only
  -in : Input filename
  -maxcenters : Generate structures from connection-table only
  -out : Output filename (.ism or .sdf format only)
```

### 5.2.2 Parameters

- in** This required parameter indicates the input molecule file.
- out** This required parameter indicates the output molecule file. The output file format is limited to isomeris SMILES (.ism NOT .smi) or sd format since these are the only formats which fully specify stereocenters without the need for three-dimensional coordinates.
- forceflip** This parameter forces flipper to modify *all* of the stereocenters in a molecule. If `-forceflip` is false (the default), flipper only enumerates stereocenters which do not already have a specified stereochemistry. [default = false]
- maxcenters** Obviously, the number of molecules generated by enumerating the stereocenters is  $2^{\hat{N}}$ , where N is the number of stereocenters. In some instances, this may be larger than is desired. The `-maxcenters` parameter indicates the maximum number of stereocenters which will be fully enumerated. If a molecule has more than `-maxcenters` stereocenters, flipper will randomly enumerate  $2^{\hat{-maxcenters}}$  instances from the full set of potential isomers. [default = 12]

### 5.2.3 Example executions

Below are three example flipper executions. Each example is followed by a brief description of the parameter's effects.

```
prompt> flipper -in drugs.ism -out enumerated.ism
```

This execution will examine each stereocenter in `drugs.ism`. If the stereocenter does not have a specified stereochemistry, `flipper` will generate molecules with each of the stereochemistry states at each stereocenter.

```
prompt> flipper -in drugs.ism -out enumerated.ism -forceflip true
```

This execution will differ from the first in that *all* stereocenters will be enumerated, regardless of whether the stereochemistry is specified in the input file.

```
prompt> flipper -in drugs.ism -out enumerated.ism -maxcenters 6
```

In this execution, `flipper` will exhaustively enumerate all unspecified stereocenters in molecules with up to 6 stereocenters. For molecules with more than 6 stereocenters, a random set of 64 isomers will be generated from the larger set of potential stereoisomers.

---

# Installation and Platform Notes

## 6.1 Licenses

To run Omega you will need to obtain a license file for Omega from OpenEye Scientific Software (support@eyesopen.com). The license file will be one of the 2 options:

1. In a file pointed to by the environment variable `$OE_LICENSE` (*preferred*).
2. Named `oe_license.txt` and must be placed in the directory specified by the environment variable `$OE_DIR`.

If you intend on running multi-processor Omega via PVM, only the master machine needs access to the license file.

## 6.2 Installation

### 6.2.1 Linux/Unix

By default, all OpenEye applications are installed in a single tree, with the latest script of each app installed in `openeye/bin`. This script determines that actual platform at run-time and calls the actual executable under the `openeye/arch` directory.

Assuming the installation is in `/usr/local/openeye`, all that is necessary to run Omega (and the other included apps) is to add `/usr/local/openeye/bin` to your `PATH`.

### 6.2.2 Windows

On Windows, we now provide a standard EXE setup installer. By default, all OpenEye apps will install into `C:\OpenEye`. In order to run Omega (and the other included apps) in a command shell or Cygwin terminal, you must add `C:\OpenEye\bin` to your `PATH` environment variable in the System Control Panel.

### 6.2.3 OS X

On OS X, we now provide a standard pkg setup installer delivered as a dmg. By default, all OpenEye apps will install into `/Applications/OpenEye`. In order to run Omega (and the other included apps) in a terminal you must add `/Applications/OpenEye/bin` to your `PATH` environment variable

## 6.3 PVM

PVM or parallel virtual machine is a freely available library for running processes on more than one processor on one or more machines. Omega can take advantage of PVM to distribute jobs over multiple processors. To do this PVM must be installed on all the machines Omega will be distributed over. The PVM source is freely available from

[http://www.csm.ornl.gov/pvm/pvm\\_home.html](http://www.csm.ornl.gov/pvm/pvm_home.html)

however many Linux distributions, and some Unix versions, include PVM by default. Omega is built with the current PVM version 3.4.4, but should also work with PVM version 3.4.3. Omega does not support PVM under Windows.

To use Omega with PVM you must do one of the following.

1. Place a link or copy of the Omega executable in `$PVM_ROOT/bin/$PVM_ARCH`
2. Define the environment variable `PVM_PATH`, which names the directory where the Omega executable resides.

The environment variables `PVM_ROOT` and `PVM_ARCH` should be defined globally as part of the PVM installation. `PVM_PATH` is generally a user-defined environment variable, and must be defined for all shells (*i.e.*, you can't just define it in the shell you're launching Omega in).

*NOTE* : There is no specific slave executable. The executable distributed for this program serves as both a master and slave PVM program as well as a single processor version.

## 6.4 PVM hosts file

After PVM and Omega are installed, each Omega job run via PVM must be supplied a file (via `-pvmconf`) that describes the hosts and number of CPU's to use. Each line in this file includes the word "host" followed by the hostname of the slave followed by the number of CPU's to use on that slave.

To use 10 slaves on a linux cluster (where nodes are named "linux1", "linux2", etc.)

```
host linux1 2
host linux2 2
host linux3 2
host linux4 2
host linux5 2
```

To use 32 CPU's on a single-image system like an SGI Altix, a single line can be used:

```
host altix 32
```

## 6.5 PVM Scaling

PVM jobs involve a certain of network traffic, especially sending multi-conformer molecules from the slaves back to the master. Omega slave jobs are relatively fast compared to the network I/O time, so for scaling beyond 32 CPUs, additional effort is required in setting up a job. For low numbers of CPUs or for divide-and-conquer style applications, using `.oeb.gz` as the output file format is fine.

As of version 2.0, Omega can write rotor-offset-compressed OEB files (via the `-roc` command-line flag) that greatly reduce the I/O on the master and allow downstream applications such as ROCS and Fred to scale to larger number of CPUs.

**Part III**

**Toolkit**

# OEOmega Classes

## 7.1 OEOmega

```
class OEOmega
```

This class represents OEOmega

### 7.1.1 Constructors

```
OEOmega(OEPlatform::oeistream *torlib=0)
```

Default and copy constructors.

### 7.1.2 operator bool

```
operator bool() const
```

### 7.1.3 operator()

```
bool operator()(OEChem::OEMCMolBase &mol,  
               const OESystem::OEUnaryPredicate< OEChem::OEAtomBase > *fixed=0)
```

### 7.1.4 AddFragLib

```
void AddFragLib(OEPlatform::oeistream &)  
void AddFragLib(const std::string &)  
void AddFragLib()
```

Add an external fraglib. This will be used in addition to the built-in fraglib. To use only this fraglib, call ClearFragLibs first.

### 7.1.5 ClearFixFile

```
void ClearFixFile()
```

Clears out the fragment that has been restrained. This allows the OEOmega object to be used in an unrestrained manner after it had previously been used with a fixed fragment.

### 7.1.6 ClearFragLibs

```
void ClearFragLibs()
```

Clear all fraglibs from the OEOmega object.

### 7.1.7 GetBuildForceField

```
std::string GetBuildForceField() const
```

Returns the text name of the forcefield currently set for building structures from 1D/2D to 3D.

### 7.1.8 GetCanonOrder

```
bool GetCanonOrder() const
```

Return the state of the canon order flag. If true, OEOmega will order atoms into canonical order before any other calculations.

### 7.1.9 GetCommentEnergy

```
bool GetCommentEnergy() const
```

Get the state of the comment energy flag. If true, OEOmega will place conformer energy into the comments that appear if the output is written to a MOL2 file.

### 7.1.10 GetDielectric

```
double GetDielectric() const
```

### 7.1.11 GetEnergyRange

```
std::vector< double > GetEnergyRange() const
```

### 7.1.12 GetEnergyWindow

```
double GetEnergyWindow() const
```

### 7.1.13 GetEnumNitrogen

```
bool GetEnumNitrogen() const
```

### 7.1.14 GetEnumRing

```
bool GetEnumRing() const
```

Get the state of the enum ring flag. If true, OEOmega will enumerate alternate ring conformations as part of the conformer search.

### 7.1.15 GetExponent

```
double GetExponent() const
```

### 7.1.16 GetFixDeleteH

```
bool GetFixDeleteH() const
```

Get the state of the fix delete H flag. If true, hydrogens will be stripped from the fix file molecule before it is used to match input structures.

### 7.1.17 GetFixMaxMatch

```
unsigned int GetFixMaxMatch() const
```

Get the state of the fix max match flag. This value controls the max number of different substructure search matches will be used. Only matters if the fixfile has more than one subsearch match in the input molecules.

### 7.1.18 GetFixMol

```
const OEChem::OEMolBase &GetFixMol() const
```

Get a reference to the internal fixmol. If you need to change it, you have to copy it out and then re-set it with SetFixMol.

### 7.1.19 GetFixRMS

```
double GetFixRMS() const
```

### 7.1.20 GetFixSubSearch

```
const OEChem::OESubSearch &GetFixSubSearch() const
```

### 7.1.21 GetFixUniqueMatch

```
bool GetFixUniqueMatch() const
```

### 7.1.22 GetFromCT

```
bool GetFromCT() const
```

### 7.1.23 GetIncludeInput

```
bool GetIncludeInput() const
```

### 7.1.24 GetMaxConfGen

```
unsigned int GetMaxConfGen() const
```

### 7.1.25 GetMaxConfRange

```
std::vector< unsigned int > GetMaxConfRange() const
```

### 7.1.26 GetMaxConfs

```
unsigned int GetMaxConfs() const
```

### 7.1.27 GetMaxRotors

```
int GetMaxRotors() const
```

### 7.1.28 GetMaxSearchTime

```
double GetMaxSearchTime() const
```

### 7.1.29 GetRMSRange

```
std::vector< double > GetRMSRange() const
```

### 7.1.30 GetRMSThreshold

```
double GetRMSThreshold() const
```

### 7.1.31 GetRangeIncrement

```
unsigned int GetRangeIncrement() const
```

### 7.1.32 GetRotorOffset

```
bool GetRotorOffset() const
```

### 7.1.33 GetSDEnergy

```
bool GetSDEnergy() const
```

### 7.1.34 GetSearchForceField

```
std::string GetSearchForceField() const
```

### 7.1.35 GetTorsionDrive

```
bool GetTorsionDrive() const
```

### 7.1.36 GetWarts

```
bool GetWarts() const
```

### 7.1.37 SetBuildForceField

```
bool SetBuildForceField(const std::string &)
```

### 7.1.38 SetCanonOrder

```
void SetCanonOrder(bool)
```

### 7.1.39 SetCommentEnergy

```
void SetCommentEnergy(bool)
```

### 7.1.40 SetDielectric

```
void SetDielectric(double dielectric)
```

### 7.1.41 SetEnergyRange

```
void SetEnergyRange(const std::string &rangeString)  
void SetEnergyRange(const std::vector< double > &)
```

### 7.1.42 SetEnergyWindow

```
void SetEnergyWindow(double)
```

#### 7.1.43 SetEnumNitrogen

```
void SetEnumNitrogen(bool)
```

#### 7.1.44 SetEnumRing

```
void SetEnumRing(bool)
```

#### 7.1.45 SetExponent

```
void SetExponent(double exponent)
```

#### 7.1.46 SetFixDeleteH

```
void SetFixDeleteH(bool)
```

#### 7.1.47 SetFixFile

```
void SetFixFile(OEChem::oemolistream &, unsigned int aopts=OEChem::OEExprOpts::DefaultAtoms, unsigned int bopt
```

#### 7.1.48 SetFixMaxMatch

```
void SetFixMaxMatch(unsigned int)
```

#### 7.1.49 SetFixMol

```
void SetFixMol(const OEChem::OEMolBase &, unsigned int aopts=OEChem::OEExprOpts::DefaultAtoms, unsigned int b
```

#### 7.1.50 SetFixQuery

```
void SetFixQuery(const OEChem::OESubSearch &)
```

#### 7.1.51 SetFixRMS

```
void SetFixRMS(double)
```

#### 7.1.52 SetFixUniqueMatch

```
void SetFixUniqueMatch(bool)
```

### 7.1.53 SetFromCT

```
void SetFromCT(bool)
```

### 7.1.54 SetIncludeInput

```
void SetIncludeInput(bool)
```

### 7.1.55 SetMaxConfGen

```
void SetMaxConfGen(unsigned int)
```

### 7.1.56 SetMaxConfRange

```
void SetMaxConfRange(const std::vector< unsigned int > &)  
void SetMaxConfRange(const std::string &rangeString)
```

### 7.1.57 SetMaxConfs

```
void SetMaxConfs(unsigned int)
```

### 7.1.58 SetMaxRotors

```
void SetMaxRotors(int)
```

### 7.1.59 SetMaxSearchTime

```
void SetMaxSearchTime(double)
```

### 7.1.60 SetRMSRange

```
void SetRMSRange(const std::vector< double > &)  
void SetRMSRange(const std::string &rangeString)
```

### 7.1.61 SetRMSThreshold

```
void SetRMSThreshold(double)
```

### 7.1.62 SetRangeIncrement

```
void SetRangeIncrement(unsigned int)
```

### 7.1.63 SetRotorOffset

```
void SetRotorOffset(bool)
```

### 7.1.64 SetRotorPredicate

```
void SetRotorPredicate(const OESystem::OEUnaryPredicate< OEChem::OEBondBase > &pred)
```

### 7.1.65 SetSDEnergy

```
void SetSDEnergy(bool)
```

### 7.1.66 SetSearchForceField

```
bool SetSearchForceField(const std::string &)
```

### 7.1.67 SetTorsionDrive

```
void SetTorsionDrive(bool)
```

### 7.1.68 SetTorsionLibrary

```
void SetTorsionLibrary(const std::string &)  
void SetTorsionLibrary(OEPlatform::oeistream &)
```

### 7.1.69 SetWarts

```
void SetWarts(bool)
```

# OEOmega Functions

## 8.1 OEFlipper

```
OESystem::OEIterBase< OEChem::OEMolBase > *OEFlipper(const OEChem::OEMolBase &mol,  
                                                    unsigned int maxcenters=12,  
                                                    bool forceFlip=false)
```

## 8.2 OEOmegaGetLibraryRelease

```
const char *OEOmegaGetLibraryRelease()
```

## 8.3 OEOmegaGetLibraryVersion

```
unsigned int OEOmegaGetLibraryVersion()
```

## 8.4 OEOmegaGetPlatform

```
const char *OEOmegaGetPlatform()
```

## 8.5 OEOmegaGetRelease

```
const char *OEOmegaGetRelease()
```

## 8.6 OEOmegaGetVersion

```
unsigned int OEOmegaGetVersion()
```

## 8.7 OEOmegaIsLicensed

```
bool OEOmegaIsLicensed(const char *feature = 0, unsigned int *expdate = 0)
```

Determine whether a valid license file is present. This function may be called without a legitimate run-time license to determine whether it is safe to call any of OEOmega's functionality.

The "features" argument can be used to check for a valid license to OEOmega along with that feature. For example, to verify that OEOmega can be used from Python:

```
if (!OEOmegaIsLicensed("python"))
    OEThrow.Warning("OEOmega is not licensed for the python feature");
```

The second argument can be used to get the expiration date of the license. This is an array of size three with the date returned as {day, month, year}. Even if the function returns false due to an expired license, the `expdate` will show that expiration date. A value of a zeroes implies that no license or date was found.

```
unsigned int expdate[3];
if (OEOmegaIsLicensed(0, expdate))
{
    OEThrow.Info("License expires: day: %d month: %d year: %d",
                expdate[0], expdate[1], expdate[2]);
}
```

## 8.8 OESliceEnsemble

```
bool OESliceEnsemble(OEChem::OEMCMolBase &mol, double rms=0.8,
                    double ewindow=10.0, unsigned maxconfs=400)
```

# Release Notes

## 9.1 Omega 2.3.2

### 9.1.1 Bug fixes

1. Fixed a bug where molecules containing boron or selenium could cause a crash. These molecules now are written to the '.fail' file due to missing force field parameters and the program will proceed to the next molecule.

## 9.2 Omega 2.3.1

### 9.2.1 New Features

1. The Omega application now includes a utility named oeb2sdconf that allows Omega generated conformers to be used with the MOE program. Please contact Chemical Computing Group support if you require assistance with this utility.

### 9.2.2 Bug fixes

1. Fixed a bug where the Omega2 PVM features would not work on some platforms.
2. Fixed a bug where the omega2 script would not function properly with filenames that had spaces in them.

## 9.3 Omega 2.3.0

### 9.3.1 New Features

1. The distribution and installation of Omega has been modified. The Windows distribution is now a standard EXE installer, and the OS X distribution is a dmg containing a standard pkg installer. The executables are now scripts that chose the correct version of the program at runtime. Please see the Installation and Platform Notes for details.
2. The defaults for MaxConfs and RMSThreshold are changed in this version. The default for MaxConfs is reduced from 400 to 200, while the value for RMSThreshold goes from 0.8 down to 0.5. These changes are based on maintaining good reproduction of the crystal structure test set, virtual screening tests with ROCS and FRED, and pose prediction tests with FRED.
3. The method `ClearFixFile` is added to the Omega library. This allows an `OEOmega` instance to be reused for an unrestrained search after it has previously been used for a fixed search.
4. Untitled molecules are given unique titles of `omega_1`, `omega_2`, and so on. Titles allow molecules to be located in the log files.

### 9.3.2 Bug fixes

1. Fixed a major bug where molecules that have missing torsion rules caused Omega to crash. Molecules with missing torsion rules are now sent to the `omega2.fail` file and are not processed. Missing torsion rules are typically caused by highly unusual molecules for which OEChem cannot properly assign valence to all atoms.
2. Fixed a bug in duplicate conformer removal that caused some conformers that should have been removed to pass through as unique. This bug also caused occurrences where a reduction in `maxconfs` led to an increase in the number of conformers returned.
3. Fixed a bug that caused SD data on single conformer molecules to be removed.
4. Fixed a bug in the Flipper library that left 3D coordinates intact, which caused stereo chemistry to be ambiguous. The Flipper library now replaces 3D coordinates with zeros.
5. Fixed a bug where the `SyM` tag from previous Omega calculations caused problems when those molecules were passed back in. Also, the `O2MolId` tag is now removed from molecules before they are sent to the output.
6. When the number of stereo centers exceeds the number of `-maxcenters` ( $M$ ), Flipper now generates  $2^M$  random non-repeating flips. Previous versions generated  $2^M$  random flips, leading to molecules with the same stereochemistry being returned multiple times.

## 9.4 Omega 2.2.2

### 9.4.1 Bug fixes

1. Fixed a major bug that would cause highly-symmetric, highly-fluorinated compounds to crash the program or hang a slave in PVM mode.
2. Fixed a bug in atom typing when building fragments from scratch that in some rare cases could result in differing fragment geometries depending on the order of input of the molecules.
3. Added more places that check for time exceeding max time. This is more just to prevent overly large molecules from appearing to hang.
4. If Omega is called with a fixed predicate, this now implies that FromCT is false. In other words, if you want to use a predicate to fix atoms, you need to use a 3D structure as input.

## 9.5 Omega 2.2.1

### 9.5.1 Bug fixes

1. The new build algorithm introduced in v2.2.0 could on rare occasions generate a structure with heavy atom clashes. This release is mainly a bug fix for this problem.

**Part IV**

**Appendices**

## Param Files

Omega's command-line interface can be efficiently run using the `-param` command line parameter followed by the name of a `.param`-file. Param files are simply files that contain one command-line parameter on each line. Every execution of `omega` generates a `.param` file called *prefix.param*. This file contains all of the parameters used by Omega. Further, this file can be used in subsequent Omega runs with the `-param` flag either with or without user modifications.

The following execution will generate a `.parm` file called "omega.parm" which is listed below. This file could be edited and used with subsequent Omega runs.

```
prompt> omega2 -in drugs.smi -out drugs.oeb.gz
```

Listing of "omega.parm":

```
#Interface settings
#File Options :
  -commentEnergy false (default)
  -in drugs.smi
  -includeInput false (default)
  #-log (Not set, no default)
  -out drugs.oeb.gz
  #-param (Not set, no default)
  -prefix omega2 (default)
  -rotorOffsetCompress true (default)
  -sdEnergy false (default)
  #-status (Not set, no default)
  -verbose false (default)
  -warts false (default)

#3D Construction Parameters :
  #-addfraglib (Not set, no default)
  -buildff mmff94s_NoEstat (default)
  -canonOrder true (default)
  -deleteFixHydrogens true (default)
  #-dielectric (Not set, no default)
  #-exponent (Not set, no default)
  #-fixfile (Not set, no default)
  -fixrms 0.150000 (default)
  -fromCT true (default)
  -maxmatch 10 (default)
  #-setfraglib (Not set, no default)
  -umatch true (default)

#Structure Enumeration :
  -enumNitrogen true (default)
  -enumRing true (default)
```

---

```
#Torsion Driving Parameters :
#-erange (Not set, no default)
-ewindow 10.000000 (default)
#-maxConfRange (Not set, no default)
-maxconfgen 50000 (default)
-maxconfs 400 (default)
-maxrot -1 (default)
-maxtime 120.000000 (default)
-rangeIncrement 5 (default)
-rms 0.800000 (default)
#-rmsrange (Not set, no default)
-searchcff mmff94s_Trunc (default)
#-torlib (Not set, no default)

#PVM Parameters :
#-pvmconf (Not set, no default)
-pvmdebug false (default)
#-pvmlog (Not set, no default)
-pvmpass 10 (default)
```

# BIBLIOGRAPHY

- [1] Jonas Boström, “Reproducing the Conformations of Protein-Bound Ligands: A Critical Evaluation of Several Popular Conformational Searching Tools”, *Journal of Computer-Aided Molecular Design (JCAMD)*, Vol. 15, pp. 1137, 2002.
- [2] J. Boström, J. R. Greenwood, and J. Gottfries, “Assessing the Performance of OMEGA with Respect to Retrieving Bioactive Conformations”, *Journal of Molecular Graphics and Modeling*, Vol. 21, pp. 449–462, 2003.
- [3] Mongoose
- [4] Wizard
- [5] J. Gasteiger, C. Rudolph and J. Sadowski, “Automatic Generation of 3D Atomic Coordinates for Organic Molecules”, *Tetrahedron Comp. Method.*, Vol 3., pp. 537-547, 1990.
- [6] J. Sadowski and J. Gasteiger, “From Atoms and Bonds to Three-dimensional Atomic Coordinates: Automatic Model Builders”, *Chemical Reviews*, Vol. 93, pp. 2567-2581, 1993.
- [7] J. Sadowski, J. Gasteiger and G. Klebe, “Comparison of Automatic Three-dimensional Model Builders using 639 X-Ray Structures”, *Journal of Chemical Information and Computer Science (JCICS)*, Vol. 34, pp. 1000-1008, 1994.
- [8] Concord
- [9] David C. Spellmeyer, A.K. Wong, M.J. Bower and J.M. Blaney, “Conformational Analysis using Distance Geometry Methods”, *Journal of Molecular Graphics and Modeling*, Vol. 15, No. 1, pp. 18-36 1997.
- [10] Matthew T. Stahl, “Rapid 3D Database Searching”, *IIR: Computational Drug Design*, circa 2000.
- [11] Matthew T. Stahl, “You want conformers? I’ll give you conformers!”, *2nd Annual OpenEye Customers, Users, and Programmers Meeting*, March 2001.
- [12] J. Andrew Grant, Anthony Nicholls, A. Geoffrey Skillman, and Matthew T. Stahl, “Dude, where are my conformers?”, *222nd National ACS Meeting*, August 25th, 2001.
- [13] Matthew T. Stahl, “Omega, AESOP, and other cautionary tales of naming programs.”, *3rd Annual OpenEye Customers, Users, and Programmers Meeting*, March 2002.
- [14] T.A. Halgren, “Merck Molecular Force Field: I. Basis, Form, Scope, Parameterization and Performance of MMFF94”, *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 490–519, 1996.

- [15] T.A. Halgren, "Merck Molecular Force Field: II. MMFF94 van der Waals and Electrostatic Parameters for Intermolecular Interactions", *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 520–552, 1996.
- [16] T.A. Halgren, "Merck Molecular Force Field: III. Molecular Geometries and Vibrational Frequencies", *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 553–586, 1996.
- [17] T.A. Halgren and R.B. Nachbar, "Merck Molecular Force Field: IV. Conformational Energies and Geometries for MMFF94", *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 687–615, 1996.
- [18] T.A. Halgren, "Merck Molecular Force Field: V. Extension of MMFF94 using Experimental Data, Additional Computational Data and Empirical Rules", *Journal of Computational Chemistry*, Vol. 17, No. 5, pp. 616–641, 1996.
- [19] T.A. Halgren, "MMFF VI. MMFF94s Option for Energy Minimization Studies", *Journal of Computational Chemistry*, Vol. 20, pp. 720–729, 1999.
- [20] T.A. Halgren, "MMFF VII. Characterization of MMFF94, MMFF94s and Other Widely Available Force Fields for Conformational Energies and for Intermolecular Interaction Energies and Geometries", *Journal of Computational Chemistry*, Vol. 20, pp. 730–748, 1999.
- [21] M. Sykes, B. Pickup, J.A. Grant, "The Sheffield Solvation Model", *In preparation*.