



OpenEye
Scientific Software

OmegaTK – Python

Release 2.4.0

OpenEye Scientific Software, Inc.

November 02, 2009

CONTENTS

1	Front Matter	1
2	Omega Theory	3
2.1	Omega Theory	3
3	Toolkit	7
3.1	Omega Examples	7
4	API	9
4.1	OEConfGen Classes	9
4.2	OEConfGen Functions	20
5	Release Notes	23
5.1	OmegaTK 2.4.0	23
5.2	OmegaTK 2.3.3	24
5.3	OmegaTK 2.3.2	24
5.4	OmegaTK 2.3.1	24
5.5	OmegaTK 2.3.0	25
5.6	OmegaTK 2.2.2	25
5.7	OmegaTK 2.2.1	26
5.8	Indices and tables	26
	Index	27

FRONT MATTER

Copyright 1997-2009 OpenEye Scientific Software, Santa Fe, New Mexico. All rights reserved.

All rights reserved. This material contains proprietary information of OpenEye Scientific Software. Use of copyright 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, OS X, 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. 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 Schrodinger, Inc.

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.

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

OMEGA THEORY

2.1 Omega 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 molecule 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 that 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 effect 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 base 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.

2.1.1 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.

2.1.2 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 flipper section in the application manual. 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 conformers with Omega.

2.1.3 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 the Makefraglib section in the application

manaul. 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.

TOOLKIT

3.1 Omega Examples

3.1.1 OEOMega Examples

The following code example is a simple example of how to generate conformers using the OEOMega object.

Listing 1: Generating Conformers

```
#!/usr/bin/env python
#####
# Copyright (C) 2006, 2007, 2008, 2009 OpenEye Scientific Software, Inc.
#####
import sys
from openeye.oechem import *
from openeye.oeomega import *

ifs=oemolistream()
ifs.open(sys.argv[1])

ofs=oemolostream()
ofs.open(sys.argv[2])

omega = OEOMega()

mol = OEMol()
while OEReadMolecule(ifs, mol):
    omega(mol)
    OEWriteMolecule(ofs, mol)
```

The following code example is a simple example of how to generate a single conformer and how to set the *OEErrorLevel* to make the library quiet.

Listing 2: Generating a Single Conformer Quietly

```
#!/usr/bin/env python
#####
# Copyright (C) 2006, 2007, 2008, 2009 OpenEye Scientific Software, Inc.
```

```
#####  
import sys  
from openeye.oechem import *  
from openeye.oeomega import *  
  
def main(argv = [__name__]):  
    if len(argv) != 3:  
        OEThrow.Usage("%s <infile> <outfile>" % argv[0])  
  
    OEThrow.SetLevel(OEErrorLevel_Info);  
  
    ifs = oemolistream(argv[1])  
    ofs = oemolostream(argv[2])  
  
    omega = OEOmega()  
    omega.SetMaxConfs(1)  
  
    for mol in ifs.GetOEMols():  
        OEThrow.Info("Title: %s" % mol.GetTitle())  
        omega(mol)  
        OEWriteMolecule(ofs, mol)  
    return 0  
  
if __name__ == "__main__":  
    sys.exit(main(sys.argv))
```

3.1.2 Flipper Examples

The following code example is a simple example of how to use the flipper. Flipper is used to generate stereoisomers. Stereoisomers should be generated before generating conformers.

Listing 3: Generating Stereoisomers

```
#!/usr/bin/env python  
#####  
# Copyright (C) 2006, 2007, 2008, 2009 OpenEye Scientific Software, Inc.  
#####  
import sys  
from openeye.oechem import *  
from openeye.oeomega import *  
  
ifs=oemolistream()  
ifs.open(sys.argv[1])  
  
inmol = OEGraphMol()  
while OEReadMolecule(ifs, inmol):  
    for mol in OEFlipper(inmol):  
        print OECreatIsoSmiString(mol),mol.GetTitle()
```

4.1 OEConfGen Classes

4.1.1 OEOmega

`class` `OEOmega`

This class represents *OEOmega*.

Constructors

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

Default and copy constructors.

operator()

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

operator bool

```
operator bool() const
```

AddFragLib

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

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

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.

ClearFragLibs

```
void ClearFragLibs()
```

Clear all fraglibs from the `OEOmega` object

GetBuildForceField

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

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

GetCanonOrder

```
bool GetCanonOrder() const
```

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

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.

GetDielectric

```
double GetDielectric() const
```

GetEnergyRange

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

GetEnergyWindow

```
double GetEnergyWindow() const
```

GetEnumNitrogen

```
bool GetEnumNitrogen() const
```

GetEnumRing

```
bool GetEnumRing() const
```

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

GetExponent

```
double GetExponent() const
```

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.

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.

GetFixMol

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

GetFixRMS

```
double GetFixRMS() const
```

GetFixSubSearch

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

GetFixUniqueMatch

```
bool GetFixUniqueMatch() const
```

GetFromCT

```
bool GetFromCT() const
```

GetIncludeInput

```
bool GetIncludeInput() const
```

GetMaxConfGen

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

This method is deprecated.

GetMaxConfRange

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

GetMaxConfs

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

GetMaxPoolSize

```
unsigned int GetMaxPoolSize() const
```

This method is deprecated.

GetMaxRotors

```
int GetMaxRotors() const
```

GetMaxSearchTime

```
double GetMaxSearchTime() const
```

GetRMSRange

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

GetRMSThreshold

```
double GetRMSThreshold() const
```

GetRangeIncrement

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

GetRotorOffset

```
bool GetRotorOffset() const
```

Returns `true` if rotor offset compress is turned on, returns `false` if it is turned off.

GetSDEnergy

```
bool GetSDEnergy() const
```

GetSearchForceField

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

GetTorLib

```
OETorLib &GetTorLib()  
const OETorLib &GetTorLib() const
```

Retrieve a reference to the `OETorLib` instance from the `OEOmega` instance.

GetTorsionDrive

```
bool GetTorsionDrive() const
```

GetWarts

```
bool GetWarts() const
```

SetBuildForceField

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

SetCanonOrder

```
void SetCanonOrder(bool)
```

SetCommentEnergy

```
void SetCommentEnergy(bool)
```

SetDielectric

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

SetEnergyRange

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

SetEnergyWindow

```
void SetEnergyWindow(double)
```

SetEnumNitrogen

```
void SetEnumNitrogen(bool)
```

SetEnumRing

```
void SetEnumRing(bool)
```

SetExponent

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

SetFixDeleteH

```
void SetFixDeleteH(bool)
```

SetFixFile

```
void SetFixFile(OEChem::oemolistream &,
               unsigned int aopts=OEChem::OEExprOpts::DefaultAtoms,
               unsigned int bopts=OEChem::OEExprOpts::DefaultBonds)
```

SetFixMaxMatch

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

SetFixMol

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

SetFixQuery

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

SetFixRMS

```
void SetFixRMS(double)
```

SetFixSmarts

```
void SetFixSmarts(const std::string &)
```

This method sets a string for SMARTS pattern matching to fix a portion of the molecule. 3D coordinates must be available from either the input file with `OEOmega.SetFromCT` set to false or from having `OEOmega.SetFixMol` set.

SetFixUniqueMatch

```
void SetFixUniqueMatch(bool)
```

SetFromCT

```
void SetFromCT(bool)
```

SetIncludeInput

```
void SetIncludeInput(bool)
```

SetMaxConfGen

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

This method is deprecated.

SetMaxConfRange

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

SetMaxConfs

```
bool SetMaxConfs(unsigned int)
```

This method sets how many confs are returned by `OEOmega.operator()`. This method returns true if the parameter is set correctly and returns false if the parameter is too high for the theoretical maximum memory available for the platform. Only the theoretical maximum memory is checked at this time – duplicate removal could be skipped during conformer generation if there is not enough actual memory.

SetMaxPoolSize

```
void SetMaxPoolSize(unsigned int)
```

This method is deprecated.

SetMaxRotors

```
void SetMaxRotors(int)
```

SetMaxSearchTime

```
void SetMaxSearchTime(double)
```

SetRMSRange

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

SetRMSThreshold

```
void SetRMSThreshold(double)
```

SetRangeIncrement

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

SetRotorOffset

```
void SetRotorOffset(bool)
```

Passing in a value of `true` will turn on rotor offset compress and passing in a value of `false` will turn it off. The default value is set to `false` for OmegaTK and set to `true` in the OMEGA application. It is safe to have this compression algorithm turned on when molecules are passed directly to `OEWriteMolecule`; however, handling the coordinates of molecules after running Omega is not safe in the toolkits so the default is `false`.

SetRotorPredicate

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

SetSDEnergy

```
void SetSDEnergy(bool)
```

SetSearchForceField

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

SetTorLib

```
bool SetTorLib(const OETorLib &torlib)
```

Sets the torsion library by passing in an `OETorLib` instance.

SetTorsionDrive

```
void SetTorsionDrive(bool)
```

SetTorsionLibrary

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

A convenience method left in for backwards compatibility. The torsion library may be set using this method instead of using the `OETorLib` object directly.

SetWarts

```
void SetWarts(bool)
```

UseStrictAtomTypes

```
UseStrictAtomTypes(bool strict=true)
```

Starting in version 2.4.0 the default is for a molecule to fail if any atom does not have a valid MMFF atom type. Calling this method with *false* will allow the atom typer to look for a different atom type of the same element and replace the failing atom type with a valid one that is “close enough”.

4.1.2 OETorLib

```
class OETorLib
```

This class represents *OETorLib*.

Constructors

```
OETorLib()
```

Default and copy constructors.

operator bool

```
operator bool() const
```

Returns `false` if the `torlib` is empty, otherwise returns `true`.

AddTorsionLibrary

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

Takes a `std::string` or an `oeistream` to add a torsion library. This will put the new torsion library above the current torsion library, which means that they will be checked first for a match. To replace the torsion library, call `OETorLib.SetTorsionLibrary`. The following is an example of a general rule. The format for rules is a SMARTS pattern followed by the torsion angles.

```
[*:1][CH2:2][a:3][a:4] 0 180 90 -90 45 -45 135 -135
```

AddTorsionRule

```
bool AddTorsionRule(const std::string &)
bool AddTorsionRule(OEChem::OEMolBase &qmol, std::vector<int> &degrees)
```

Add a single torsion rule. The rule can either be passed in as a `std::string` with the typically format or as an `OEMolBase` with a corresponding `std::vector<int>` containing the angles.

ClearTorsionLibrary

```
void ClearTorsionLibrary()
```

Clear the current torsion library. Omega cannot operate with an empty torsion library, so a new torsion library must be set before running Omega.

GetTorRules

```
OESystem::OEIterBase<OEChem::OESubSearch> *GetTorRules()
```

Returns an iterator over the torsion rules. The rules are stored as `OESubSearch` objects.

ResetTorsionLibrary

```
void ResetTorsionLibrary()
```

Resets the torsion library stored internally by Omega.

SetTorsionLibrary

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

Replaces the current torsion library with the one passed in as an argument.

4.2 OEConfGen Functions

4.2.1 OEFlipper

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

4.2.2 OEOmegaGetArch

```
const char *OEOmegaGetArch()
```

4.2.3 OEOmegaGetLibraryRelease

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

4.2.4 OEOmegaGetLibraryVersion

```
unsigned int OEOmegaGetLibraryVersion()
```

4.2.5 OEOmegaGetLicensee

```
bool OEOmegaGetLicensee(std::string &licensee)
```

4.2.6 OEOmegaGetPlatform

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

4.2.7 OEOmegaGetRelease

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

4.2.8 OEOmegaGetSite

```
bool OEOmegaGetSite(std::string &site)
```

4.2.9 OEOmegaGetVersion

```
unsigned int OEOmegaGetVersion()
```

4.2.10 OEOmegaIsLicensed

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

4.2.11 OESliceEnsemble

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

RELEASE NOTES

5.1 OmegaTK 2.4.0

5.1.1 New features

- The `MaxConfGen` and `MaxPoolSize` variables are now set internally and adjusted automatically subject to the user defined variable of `MaxConfs`. Additionally, hard limits have been removed from these variables and they are now only limited by available memory. The following methods are deprecated: `OEOmega.SetMaxConfGen`, `OEOmega.GetMaxConfGen`, `OEOmega.SetMaxPoolSize`, `OEOmega.GetMaxPoolSize`
- The method `OEOmega.SetMaxConfs` has been changed from *void* to *bool*. The method will return *true* if the parameter has been set properly and will return *false* if the number is too high and would cause the memory required for duplicate removal to exceed the maximum theoretical memory for the platform.
- The method `OEOmega.SetFixSmarts` has been created to allow a smarts pattern to be used to fix a portion of the molecule. This requires 3D coordinates to be available from either a 3D input file with `OEOmega.SetFromCT` set to *false* or from a molecule set using `OEOmega.SetFixMol`.
- Fragment generation for `OEOmega` is now the same as the `makefraglib` application. Previous versions used a faster but more approximate version of fragment generation.
- The `OEOmega` default has been changed to no longer use “close enough” atom typing. If a proper MMFF atom type cannot be found for an atom then the molecule will fail. Previous versions would use a different atom type of the same element if it was available. The previous behavior of atom type substitution can be enabled by calling `OEOmega.UseStrictAtomTypes` with the argument *false*.
- The error level for most messages coming from the OmegaTK library has been lowered to *Verbose*. The library can be run in a much quieter manner if the error level is set to *Info* by the user.

5.1.2 Bug fixes

- Crashes have been fixed that could occur when the `FixFile` could not find a valid match or did not have enough atoms to match against in a ring system.
- Program no longer crash if there is not enough memory for duplicate removal. If there is not enough memory available for allocation then duplicate removal will be skipped for the molecule and `OEOmega.operator()` will return *true*.

5.2 OmegaTK 2.3.3

5.2.1 New features

- New API points have been added: `OETorLib.ResetTorsionLibrary`, `OETorLib.ClearTorsionLibrary`, and `OETorLib.AddTorsionRule`. The torsion rules may be created by passing in a `OEQMolBase` and `vector<int>` of angles, or by passing in a string with the standard rule format.
- The corresponding atom symbols, bond symbols, and torsion angles are now printed on a line under the matching smarts pattern.

5.2.2 Bug fixes

- Rotor offset compression is now set to `false` by default. Coordinate changes made in the toolkits after an Omega calculation are lost if rotor offset compression is set to `true`. If molecules are sent to `OEWriteMolecule` directly after an Omega run then rotor offset compression may safely be set to `true`, which will significantly reduce the filesize of the output.
- The energy window set by `OEOmega.SetEnergyRange` and `OEOmega.SetRangeIncrement` is now used consistently. Previous versions of Omega did not use this value properly for all aspects of an Omega calculation.

5.3 OmegaTK 2.3.2

5.3.1 Bug fixes

- 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.

5.4 OmegaTK 2.3.1

5.4.1 New features

- 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.

5.4.2 Bug fixes

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

5.5 OmegaTK 2.3.0

5.5.1 New features

- 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.
- 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.
- The method `OEOmega.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.
- Untitled molecules are given unique titles of `omega_1`, `omega_2`, and so on. Titles allow molecules to be located in the log files.

5.5.2 Bug fixes

- 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.
- 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.
- Fixed a bug that caused SD data on single conformer molecules to be removed.
- 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.
- 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.
- 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.

5.6 OmegaTK 2.2.2

5.6.1 Bug fixes

- Fixed a major bug that would cause highly-symmetric, highly-fluorinated compounds to crash the program or hang a slave in PVM mode.
- 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.
- Added more places that check for time exceeding max time. This is more just to prevent overly large molecules from appearing to hang.

- 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.

5.7 OmegaTK 2.2.1

5.7.1 Bug fixes

- 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.

5.8 Indices and tables

- *Index*
- *Search Page*

INDEX

A

AddFragLib
 OEConfGen::OEOmega, 9
AddTorsionLibrary
 OEConfGen::OETorLib, 19
AddTorsionRule
 OEConfGen::OETorLib, 19

C

ClearFixFile
 OEConfGen::OEOmega, 10
ClearFragLibs
 OEConfGen::OEOmega, 10
ClearTorsionLibrary
 OEConfGen::OETorLib, 19
Constructors
 OEConfGen::OEOmega, 9
 OEConfGen::OETorLib, 18

E

Example Code
 example1.py, 7
 flipper.py, 8
 simpleOmicron.py, 7
example1.py
 Example Code, 7

F

flipper.py
 Example Code, 8

G

GetBuildForceField
 OEConfGen::OEOmega, 10
GetCanonOrder
 OEConfGen::OEOmega, 10
GetCommentEnergy
 OEConfGen::OEOmega, 10
GetDielectric
 OEConfGen::OEOmega, 10
GetEnergyRange

 OEConfGen::OEOmega, 10
GetEnergyWindow
 OEConfGen::OEOmega, 11
GetEnumNitrogen
 OEConfGen::OEOmega, 11
GetEnumRing
 OEConfGen::OEOmega, 11
GetExponent
 OEConfGen::OEOmega, 11
GetFixDeleteH
 OEConfGen::OEOmega, 11
GetFixMaxMatch
 OEConfGen::OEOmega, 11
GetFixMol
 OEConfGen::OEOmega, 11
GetFixRMS
 OEConfGen::OEOmega, 11
GetFixSubSearch
 OEConfGen::OEOmega, 12
GetFixUniqueMatch
 OEConfGen::OEOmega, 12
GetFromCT
 OEConfGen::OEOmega, 12
GetIncludeInput
 OEConfGen::OEOmega, 12
GetMaxConfGen
 OEConfGen::OEOmega, 12
GetMaxConfRange
 OEConfGen::OEOmega, 12
GetMaxConfs
 OEConfGen::OEOmega, 12
GetMaxPoolSize
 OEConfGen::OEOmega, 12
GetMaxRotors
 OEConfGen::OEOmega, 12
GetMaxSearchTime
 OEConfGen::OEOmega, 13
GetRangeIncrement
 OEConfGen::OEOmega, 13
GetRMSRange
 OEConfGen::OEOmega, 13
GetRMSThreshold

OEConfGen::OEOmega, 13
 GetRotorOffset
 OEConfGen::OEOmega, 13
 GetSDEnergy
 OEConfGen::OEOmega, 13
 GetSearchForceField
 OEConfGen::OEOmega, 13
 GetTorLib
 OEConfGen::OEOmega, 13
 GetTorRules
 OEConfGen::OETorLib, 19
 GetTorsionDrive
 OEConfGen::OEOmega, 13
 GetWarts
 OEConfGen::OEOmega, 14

O

OEConfGen::OEFlipper, 20
 OEConfGen::OEOmega, 9
 AddFragLib, 9
 ClearFixFile, 10
 ClearFragLibs, 10
 Constructors, 9
 GetBuildForceField, 10
 GetCanonOrder, 10
 GetCommentEnergy, 10
 GetDielectric, 10
 GetEnergyRange, 10
 GetEnergyWindow, 11
 GetEnumNitrogen, 11
 GetEnumRing, 11
 GetExponent, 11
 GetFixDeleteH, 11
 GetFixMaxMatch, 11
 GetFixMol, 11
 GetFixRMS, 11
 GetFixSubSearch, 12
 GetFixUniqueMatch, 12
 GetFromCT, 12
 GetIncludeInput, 12
 GetMaxConfGen, 12
 GetMaxConfRange, 12
 GetMaxConfs, 12
 GetMaxPoolSize, 12
 GetMaxRotors, 12
 GetMaxSearchTime, 13
 GetRangeIncrement, 13
 GetRMSRange, 13
 GetRMSThreshold, 13
 GetRotorOffset, 13
 GetSDEnergy, 13
 GetSearchForceField, 13
 GetTorLib, 13
 GetTorsionDrive, 13

GetWarts, 14
 operator bool, 9
 operator(), 9
 SetBuildForceField, 14
 SetCanonOrder, 14
 SetCommentEnergy, 14
 SetDielectric, 14
 SetEnergyRange, 14
 SetEnergyWindow, 14
 SetEnumNitrogen, 14
 SetEnumRing, 14
 SetExponent, 15
 SetFixDeleteH, 15
 SetFixFile, 15
 SetFixMaxMatch, 15
 SetFixMol, 15
 SetFixQuery, 15
 SetFixRMS, 15
 SetFixSmarts, 15
 SetFixUniqueMatch, 16
 SetFromCT, 16
 SetIncludeInput, 16
 SetMaxConfGen, 16
 SetMaxConfRange, 16
 SetMaxConfs, 16
 SetMaxPoolSize, 16
 SetMaxRotors, 16
 SetMaxSearchTime, 17
 SetRangeIncrement, 17
 SetRMSRange, 17
 SetRMSThreshold, 17
 SetRotorOffset, 17
 SetRotorPredicate, 17
 SetSDEnergy, 17
 SetSearchForceField, 17
 SetTorLib, 18
 SetTorsionDrive, 18
 SetTorsionLibrary, 18
 SetWarts, 18
 UseStrictAtomTypes, 18
 OEConfGen::OEOmegaGetArch, 20
 OEConfGen::OEOmegaGetLibraryRelease, 20
 OEConfGen::OEOmegaGetLibraryVersion, 20
 OEConfGen::OEOmegaGetLicensee, 20
 OEConfGen::OEOmegaGetPlatform, 20
 OEConfGen::OEOmegaGetRelease, 20
 OEConfGen::OEOmegaGetSite, 21
 OEConfGen::OEOmegaGetVersion, 21
 OEConfGen::OEOmegaIsLicensed, 21
 OEConfGen::OESliceEnsemble, 21
 OEConfGen::OETorLib, 18
 AddTorsionLibrary, 19
 AddTorsionRule, 19
 ClearTorsionLibrary, 19

Constructors, 18
 GetTorRules, 19
 operator bool, 19
 ResetTorsionLibrary, 19
 SetTorsionLibrary, 20
 operator bool
 OEConfGen::OEOmega, 9
 OEConfGen::OETorLib, 19
 operator()
 OEConfGen::OEOmega, 9

R

ResetTorsionLibrary
 OEConfGen::OETorLib, 19

S

SetBuildForceField
 OEConfGen::OEOmega, 14
 SetCanonOrder
 OEConfGen::OEOmega, 14
 SetCommentEnergy
 OEConfGen::OEOmega, 14
 SetDielectric
 OEConfGen::OEOmega, 14
 SetEnergyRange
 OEConfGen::OEOmega, 14
 SetEnergyWindow
 OEConfGen::OEOmega, 14
 SetEnumNitrogen
 OEConfGen::OEOmega, 14
 SetEnumRing
 OEConfGen::OEOmega, 14
 SetExponent
 OEConfGen::OEOmega, 15
 SetFixDeleteH
 OEConfGen::OEOmega, 15
 SetFixFile
 OEConfGen::OEOmega, 15
 SetFixMaxMatch
 OEConfGen::OEOmega, 15
 SetFixMol
 OEConfGen::OEOmega, 15
 SetFixQuery
 OEConfGen::OEOmega, 15
 SetFixRMS
 OEConfGen::OEOmega, 15
 SetFixSmarts
 OEConfGen::OEOmega, 15
 SetFixUniqueMatch
 OEConfGen::OEOmega, 16
 SetFromCT
 OEConfGen::OEOmega, 16
 SetIncludeInput
 OEConfGen::OEOmega, 16

SetMaxConfGen
 OEConfGen::OEOmega, 16
 SetMaxConfRange
 OEConfGen::OEOmega, 16
 SetMaxConfs
 OEConfGen::OEOmega, 16
 SetMaxPoolSize
 OEConfGen::OEOmega, 16
 SetMaxRotors
 OEConfGen::OEOmega, 16
 SetMaxSearchTime
 OEConfGen::OEOmega, 17
 SetRangeIncrement
 OEConfGen::OEOmega, 17
 SetRMSRange
 OEConfGen::OEOmega, 17
 SetRMSThreshold
 OEConfGen::OEOmega, 17
 SetRotorOffset
 OEConfGen::OEOmega, 17
 SetRotorPredicate
 OEConfGen::OEOmega, 17
 SetSDEnergy
 OEConfGen::OEOmega, 17
 SetSearchForceField
 OEConfGen::OEOmega, 17
 SetTorLib
 OEConfGen::OEOmega, 18
 SetTorsionDrive
 OEConfGen::OEOmega, 18
 SetTorsionLibrary
 OEConfGen::OEOmega, 18
 OEConfGen::OETorLib, 20
 SetWarts
 OEConfGen::OEOmega, 18
 simpleOmicron.py
 Example Code, 7

U

UseStrictAtomTypes
 OEConfGen::OEOmega, 18