
SZYBKI

version 1.4.0

OpenEye Scientific Software, Inc.

October 28, 2009

9 Bisbee Ct, Suite D
Santa Fe, NM 87508
www.eyesopen.com
support@eyesopen.com

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 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

I Introduction	1
1 Overview	2
2 Theory	3
2.1 Force Field	3
II Application	6
3 Using Szybki	7
3.1 Help	7
3.2 Command line options	8
3.3 Output	14
III Toolkit	17
4 C++ Examples	18
4.1 Simple optimization of molecule(s) in vacuum	18
4.2 Simple minimization of molecule(s) inside the protein	19
4.3 Optimization of a set of molecules in vacuum or solution	19
4.4 Optimization of a set of molecules inside rigid protein receptor	20
4.5 Optimization of a set of molecules inside partially flexible protein receptor	21
5 OESzybki Classes and Member Functions	22
5.1 OESzybki	22
5.2 OESzybkiResults	30
6 OESzybki Functions	34
6.1 OEGetEnergyTermName	34
6.2 OESzybkiGetRelease	34
6.3 OESzybkiGetVersion	34
6.4 OESzybkiIsLicensed	34
7 OESzybki Namespaces	36
7.1 OEPotentialTerms	36
7.2 OEForceFieldType	37
7.3 OERunType	37

7.4	OESolventModel	37
7.5	OEProteinElectrostatics	38
7.6	EOptType	38
8	Release Notes	39
8.1	Szybki 1.4.0	39
8.2	Szybki 1.3.4	40
8.3	Szybki 1.3.3	40
8.4	Szybki 1.3.2	40
8.5	Szybki 1.3.1	41
8.6	Szybki 1.3.0	41
8.7	Szybki 1.2.2	42
	Bibliography	43

Part I

Introduction

Overview

OpenEye Scientific Software provides toolkit and application versions of Szybki which can be used to optimize molecules with the Merck Molecular Force Field (MMFF), by Tom Halgren (*Halgren 1996, Halgren 1999*) both in vacuum, solution and within a receptor (protein).

The Szybki toolkit API can be used for the fast creation of custom applications. It removes the need for creation and handling of adaptors and optimizer objects.

Both toolkit and application products provide several choices regarding the coordinate space in which molecules are optimized (Cartesian, torsional, translational/rotational). It is also possible to reduce the number of degrees of freedom by fixing a selected group of atoms in space and to constrain groups of atoms with a harmonic, flat-bottomed potential. Another feature is the choice of several solvation models (Poisson-Boltzmann, Sheffield, Cavity). Optimization of the ligand inside a protein receptor can be performed for a rigid or a partially flexible (side chains, polar hydrogens) protein in the vicinity of a ligand. Different models of protein-ligand electrostatic interaction can be selected.

The Szybki application can be run in parallel using PVM.

Theory

2.1 Force Field

The MMFF potential expression is:

$$V_{MMFF} = \sum_b V_b + \sum_a V_a + \sum_s V_s + \sum_o V_o + \sum_t V_t + \sum_v V_v + \sum_c V_c \quad (2.1)$$

where the seven terms respectively describe bond stretching (*b*), angle bending (*a*), stretch-bending (*s*), out-of-plane bending (*o*), torsion (*t*), Van der Waals (*v*) and electrostatic (*c*) interactions. Their functional forms are given below.

2.1.1 Bond stretching

For a bond *b* between atoms *i* and *j* the stretching potential is:

$$V_b = 143.9325 \frac{k_{ij}}{2} \Delta r_{ij}^2 (1 + c_s^2 \Delta r_{ij} + 7/12 c^2 \Delta r_{ij}^2) \quad (2.2)$$

where k_{ij} is the force constant in $md/\text{\AA}$, Δr_{ij} is the difference between actual and reference bond lengths, and c_s is so called “cubic-stretch” constant for which the value is $-2/\text{\AA}^{-1}$.

2.1.2 Angle bending

The bending potential of a bond angle *a* made by the bonds between atoms *i*, *j* and atoms *j*, *k* is given by:

$$V_a = 0.043844 \frac{k_{ijk}}{2} \Delta \vartheta_{ijk}^2 (1 + c_b \Delta \vartheta_{ijk}) \quad (2.3)$$

where k_{ijk} is the force constant in $md\text{\AA}/rad^2$, $\Delta \vartheta_{ijk}$ is the difference between actual and reference bond angles, and c_b is the so called “cubic-bend” constant for which the value is $-0.4rad^{-1}$.

2.1.3 Stretch-bend interaction

The coupling between the stretching potential of two bonds forming a bond angle and bending that angle is described by:

$$V_s = 2.5121(k_{ijk}\Delta r_{ij} + k_{kji}\Delta r_{kj})\vartheta_{ijk} \quad (2.4)$$

where k_{ijk} and k_{kji} are force constants in md/rad which couple stretches of $i-j$ and $k-j$ to the $i-j-k$ bend respectively. Δr and ϑ are defined above.

2.1.4 Out-of-plane bending

For a trigonal center j , the potential of displacement for an atom l bonded to atom j out of plane $i-j-k$ is:

$$V_o = 0.043844 \frac{k_{ijkl}}{2} \chi_{ijkl}^2 \quad (2.5)$$

where k_{ijkl} is the force constant in $md\text{\AA}/rad^2$ and χ_{ijkl} is an angle formed by the bond $j-l$ and the plane $i-j-k$.

2.1.5 Torsion interactions

For every four bonded atoms $i-j-k-l$ the torsion interaction is described by the term:

$$V_t = 0.5(V_1(1 + \cos \Phi) + V_2(1 - \cos 2\Phi) + V_3(1 + \cos 3\Phi)) \quad (2.6)$$

where V_1 , V_2 and V_3 are constants depending on atoms i, j, k, l , and Φ is the dihedral angle formed by bonds $i-j$ and $k-l$.

2.1.6 Van der Waals interactions

For a pair of atoms i and j separated by three or more bonds, where the distance between them is r_{ij} , MMFF adopts the following Van der Waals potential:

$$V_v = \epsilon_{ij} \left(\frac{1.07R_{ij}}{r_{ij} + 0.07R_{ij}} \right)^7 \left(\frac{1.12R_{ij}^7}{r_{ij}^7 + 0.12R_{ij}} - 2 \right) \quad (2.7)$$

where R_{ij} and ϵ_{ij} are defined as follows:

$$R_i = A_i \alpha_i^{0.25} \quad (2.8)$$

$$R_{ij} = 0.5(R_i + R_j)(1 + B(1 - \exp(-12\gamma_{ij}^2))) \quad (2.9)$$

$$\gamma_{ij} = (R_i - R_j)/(R_i + R_j) \quad (2.10)$$

$$\epsilon_{ij} = \frac{181.16G_iG_j\alpha_i\alpha_j}{(\alpha_i/N_i)^{0.5} + (\alpha_j/N_j)^{0.5}} \frac{1}{R_{ij}^6} \quad (2.11)$$

where α_i is atomic polarizability of atom i , B is 0.2 or 0.0 if one of the atoms is polar hydrogen, N_i and N_j are the Slater-Kirkwood effective numbers of valence electrons, G_i , G_j and A_i are scale factors.

2.1.7 Electrostatic interactions

The electrostatic interaction between two charged atoms i and j separated by at least three bonds is calculated from the standard Coulombic expression:

$$V_c = f \frac{332.0716 q_i q_j}{D(r_{ij} + \delta)} \quad (2.12)$$

where D is the dielectric constant for which the default value is 1, q_i and q_j are the MMFF partial charges on atoms i and j , r_{ij} is the interatomic distance, δ is the “electrostatic buffering” constant of 0.05\AA . Scaling factor f is 0.75 for 1,4 interactions, and 1.0 otherwise.

Part II

Application

Using Szybki

3.1 Help

Running Szybki from a command line without any options generates the banner followed by:

No argument specified on the command line

For more help type:

```
szybki --help
```

Running:

```
szybki --help all
```

generates a list of parameters and their meaning:

Complete parameter list

- param : A parameter file
- pvmconf : A text file specifying a PVM configuration
- silent : Suppress writing of processed molecule names

File Options

- complex : Name of the input protein file with ligand(s)
- fix_file : Name of the input file with atom numbers to be fixed
- heavy_rms : Reporting heavy atoms RMSD will replace all atoms RMSD
- ligands : Name of the input molecule file
- loadPG : Name of the PB or Coulomb potential grid file to be read
- log : Name of the log file prefix
- out : Name of the output file containing an optimised molecule
- out_protein : Name of the file where the coordinates of the partially optimized protein will be saved
- prefix : Replaces the default prefix for log, status, param and rpt files
- protein : Name of the input protein file
- report : Tab separated energy component output file will be created
- reportFile : Name of report (.rtp) file which replaces default name
- savePG : Name of the PB or Coulomb potential grid file to be written
- sdtag : Energies will be written as a tag in the output SD file
- verbose : Print initial/optimization/final data in single-processor mode

Potential Function Options

- MMFF94s : MMFF94s parameter set will be used
- harm_constr1 : Flat-bottom quadratic constraint force constant
- harm_constr2 : Flat-bottom constraining distance
- harm_smarts : SMARTS pattern of atoms to be harmonically constrained
- mod_vdw : VdW attractive forces will be ignored
- mol2charges : Charges for protein-ligand interaction and PB solvation will be read from the mol2 input file
- neglect_frozen : Frozen potential terms will not be calculated
- noCoulomb : Coulomb terms excluded from MMFF potential
- prot_dielectric : Dielectric constant used for protein electrostatics
- protein_elec : Protein-ligand electrostatic potential model
- protein_vdw : Cutoff distance for protein-ligand VdW potential
- shefA : A parameter in Sheffield model
- shefB : B parameter in Sheffield model
- sheffield : Sheffield solvation model will be used
- solventCA : Surface area solvation term will be used with the specified factor
- solventPB : Poisson-Boltzman solvation model will be used

Optimization Options

- conj : Default bfgs optimization is replaced by conjugate gradient method
- fix_smarts : SMARTS pattern of atoms to be fixed
- grad_conv : RMS gradient tolerance
- largest_part : Only the largest fragment of noncovalent complex kept
- max_iter : Maximum number of optimization iterations
- no_opt : Single point calculation
- opt_cart : Optimization of atomic Cartesian coordinates
- opt_solid : Optimization of a rigid ligand inside the protein
- opt_torsions : Optimization of free rotor torsions
- polarH : Optimization of protein polar hydrogens in the ligand neighborhood
- residue : Optimization of protein residues in the ligand neighborhood
- sideC : Optimization of protein residues side chains in the ligand neighborhood
- strip_water : Water molecules from the input protein will be removed

3.2 Command line options

Any order of arguments is allowed, however numeric parameters must follow the options associated with them. A more detailed description of the command line options is provided below.

3.2.1 File Options

- **complex fn**
Input file with 3D coordinates of a protein-ligand(s) complex in any format supported by OEChem. All ligands contained in the molecule file “fn” will be optimized one by one in the presence of all others. If in addition the option “-ligands” is used, all external ligands given with that option will be optimized as well. If there is spatial overlap with the internal ligand from the fn file, the internal ligand will be removed.
- **fix_file fn**
File fn should contain a list of molecule names followed by atom numbers to be fixed, e.g:
molecule1
0
10
molecule2
21
will fix atoms 0 and 10 in molecule1 and atom 21 in molecule2.
- **heavy_rms**
Default all atoms RMSD between the initial and optimized structures is replaced by heavy atoms RMSD.
- **ligands fn**
Molecular input file name containing 3D coordinates in any format supported by OEChem. In order to comply with the previous Szybki releases, an alias “-i” can be used instead of “-ligands”. This option might be used without the “-ligands” or “-i” keys but only when used last or next to last when the last option on the command line is “-out”. Warning: File name: “szybki_out.mol2” is used as the default output file name, so should be avoided as an input file name.
- **loadPG fn**
In the case when the electrostatic component of the protein-ligand interaction energy has been pre-calculated on a grid, this option forces Szybki to read the grid potential from the file fn, and use it for ligand optimization inside the protein.
- **log pre**
Prefix of the log file name, “pre.log”. If omitted, the prefix “szybki” is used by default. This option is aliased to “-l”.
- **out fn**
Output file name, in any format supported by OEChem, for an optimized ligand. If not specified, “szybki_out.mol2” or “prefix_out.mol2” (when “-prefix” is used) will be generated. Alias “-o” can be used instead of “-out”. When options “-ligands” and “-complex” are used together, internal ligands from the protein-ligand complex will be output first. Can be used without the “-out” key when it is last on the command line just after the “-ligands” (or “-i”) option.
- **out_protein fn**
Partially optimized protein will be saved in a file named fn.
- **param**
Command line options will be read from the specified file. This file may have been generated from a previous run or may be constructed *de novo*. The default name of the file is “szybki.param”. Any parameter in the setup file is superseded by the parameter on the command line. For example running “szybki -param szybki.param -i my.pdb” will perform calculations for the molecule “my.pdb” while using all other parameters taken from “szybki.param”.
- **prefix pn**
Replaces “szybki” prefix in .log, .rpt (report), .status, .param and _out.mol2 files, with the input string pn.

- **protein fn**
Protein input file with 3D coordinates in any format supported by OEChem. Usually works in combination with the “-ligands” option. No removal of pre-existing ligand in file fn will be done as in the case of the “-complex” option. When used without “-ligands” option, two scenarios will be followed depending on the presence of ligands in the file fn: If ligands are present, all of them will be optimized as in the case of “-complex” option. If not, the optimization of the entire protein will be attempted. The latter is not recommended however, because for larger proteins extreme memory usage can occur causing crashes. This option can be aliased with “-p” for compatibility with previous Szybki releases.
- **report**
An output file in tabular form of final energy components for every molecule/conformer will be generated. The name of the file is fixed as: “prefix.rpt” where “prefix” is “szybki” by default or determined by “-prefix” option.
- **reportFile fn**
Sets user selected file name, fn, for the tab file. Supersedes the “-report” option described above.
- **savePG fn**
Saves potential grid in the file named fn. Allows a significant saving in CPU time for runs when PB or Coulomb grids are used to optimize a number of ligands inside the same protein.
- **sdtag string**
In the case when the output file is in SD format an energy tag can be added. By default the final total optimized energy is included. The following potential components can be used, exactly as shown:
 1. “MMFF VdW”
 2. “MMFF Coulomb”
 3. “MMFF Bond”
 4. “MMFF Bend”
 5. “MMFF StretchBend”
 6. “MMFF Torsion”
 7. “MMFF Improper Torsion”
 8. “L MMFF VdW”
 9. “L MMFF Coulomb”
 10. “L MMFF Bond”
 11. “L MMFF Bend”
 12. “L MMFF StretchBend”
 13. “L MMFF Torsion”
 14. “L MMFF Improper Torsion”
 15. “P MMFF VdW”
 16. “P MMFF Coulomb”
 17. “P MMFF Bond”
 18. “P MMFF Bend”
 19. “P MMFF StretchBend”
 20. “P MMFF Torsion”
 21. “P MMFF Improper Torsion”
 22. “PL MMFF VdW”

23. “PL MMFF Coulomb”
24. “Ligand-Protein Energy”
25. “Sheffield Solvation”
26. “Constraint Potential”
27. “PB Solvent”
28. “Area Solvent”
29. “ __VdW”
30. “ __Coulomb”
31. “ __Protein desolv”
32. “ __Ligand desolv”
33. “ __Solvent screening”
34. “ __Grid Coulomb”
35. “ __Exact Coulomb”

The default for sdtag is “Total_energy” and is added to the SD output file in the case when “-sdtag” is not specified. If parameter “string” does not correspond to one of the available energy terms, no tag will be written. Tags starting with L, P, PL and __ are:

- L ligand intra-molecular terms only
 - P protein intra-molecular terms only
 - PL protein-ligand inter-molecular terms only
 - __ protein-ligand interaction terms
- verbose
An extensive general output will be generated containing initial and final energy and gradient data, as well as an optimization report. This option is available only for single-processor mode.

3.2.2 Potential Function Options

- MMFF94s
Modified set of MMFF94 parameters developed in 1999 by Halgren (*Halgren 1999*) will be used.
- harm_constr1 k
Constrained potential of the form: kr^2 will be imposed on all heavy atoms, where k is the force constant.
- harm_constr2 d
Constrained potential of the form: $k(r - d)^2$ will be imposed on all heavy atoms, where d is the constraining distance. Can be used only together with harm_constr1
- harm_smarts p
All atoms which belong to SMARTS pattern p will be constrained. For example input parameter “-harm_smarts cO” will result in constraining all aromatic carbon atoms and oxygen atoms bonded to them. Must be used in conjunction with “-harm_constr1”.

- mod_vdw

Regular MMFF VdW given by equation 2.7 will be replaced with:

$$V_{vdw} = \begin{cases} \epsilon_{ij} \left(\frac{1.07R_{ij}}{r_{ij}+0.07R_{ij}} \right)^7 \left(\frac{1.12R_{ij}^7}{r_{ij}^7+0.12R_{ij}} - 2 \right) & \text{for } r_{ij} < R_{ij} \\ -\epsilon_{ij} & \text{for } r_{ij} \geq R_{ij} \end{cases}$$

in which no attractive VdW forces are present. This type of VdW potential prevents so called “hydrophobic collapse”.

- mol2charges

During optimization of molecules in solution with the use of the Poisson-Boltzmann solvation model, free energy of solvation and solvent forces can be calculated with the use of atomic partial charges other than MMFF. Option “-mol2charges” allows the use of partial charges read from the input mol2 file. In the case where the ligand is optimized inside a protein, protein-ligand electrostatic interactions (Coulomb and/or Poisson-Boltzmann) could be calculated based on ligand and protein partial charges read from the mol2 input file(s). When input partial charges are not found, the MMFF94 partial charges will be used.

- neglect_frozen

When a subset of atoms is frozen during optimization, by default at the end of the optimization process a complete energy of a molecule which includes constant terms is calculated. For large molecules like proteins this could be prohibitively expensive. The usage of the “-neglect_frozen” option allows to eliminate this calculation.

- noCoulomb

Electrostatic terms defined in section 2.12 will be excluded from the force field potential. This option might be useful to prevent generation of folded structures.

- protein_dielectric *d*

The default value for the protein dielectric constant of 1.0 can be changed to a user selected value *d*.

- protein_elec *m*

This option provides 5 choices for calculating protein-ligand interaction energies. In all cases (*m* = 0-4) the MMFF VdW potential is used. Option 0 eliminates electrostatic interactions. Options 2 and 3 result in the usage of MMFF Coulomb potential on the grid (*m* = 2) or exact (*m* = 3). Options 1 and 4 provide a more realistic potential calculated according to the Poisson-Boltzmann (PB) model on the grid (*m* = 1) and at every iteration step (*m* = 4). PB options require substantially higher CPU time, particularly for large proteins. By default *m* = 0.

- protein_vdw *r*

Calculation of VdW protein-ligand interaction energy will be limited to a sphere of radius *r*. The default value is 18.0Å. In many applications a value as small as 10Å can be used with essentially no effect on the final optimized ligand geometry.

- shefA *a*

parameter *a* in the Sheffield solvation potential given in option “sheffield”. The default value is: 1.553149 (Grant 2006).

- shefB *b*

parameter *b* in the Sheffield solvation potential given in option “sheffield”. The default value is: 0.735694 (Grant 2006).

- sheffield

Usage of this option will result in adding additional electrostatic terms of the form: $-q_i q_j / \sqrt{(ar^2 + bR_i R_j)}$ where q_i , q_j are partial charges and R_i , R_j are Van der Waals radii of atoms *i* and *j* in order to mimic the solution environment (Grant 2006). This option might help to preserve unfolded structures during optimization.

- solventCA *s*
This option can be used only in combination with “solventPB” or “sheffield”. It causes inclusion of a molecular surface solvation term (called sometimes cavity solvation term) in the total energy. The recommended value of parameter *s* is 0.025. This option can be aliased with “-solventMA”, for compatibility with previous releases.
- solventPB
For optimization of small molecules in solution, the electrostatic part of molecule-solvent interactions will be calculated using Poisson-Boltzmann model.

3.2.3 Optimization Options

- conj
Conjugate gradient optimization will replace the default quasi-Newton BFGS method.
- fix_smarts *p*
All atoms which belong to SMARTS pattern *p* will be fixed. For example input parameter “-fix_smarts cN” will result in fixing all aromatic carbon atoms and amino nitrogen atoms bonded to them.
- grad_conv *c*
Optimization is terminated when the rms gradient reaches the input value *c* unless is finished earlier because of other reasons. When omitted the default convergence criteria is 0.1 on gradient vector norm: $\sqrt{\sum_i g_i g_i}$.
- largest_part
Calculations are performed only for the largest fragment of the noncovalent input complex. For example when the input file contains coordinates for the salt, [Large_cation⁺] · [Cl⁻], the use of the “-largest_part” will cause the Cl⁻ anion to be ignored. By default calculations are done for the entire complex.
- max_iter *m*
Optimization will be terminated when the number of iteration cycles reaches input number *m*.
- no_opt
A single point calculation will be performed *i.e.* no optimization of ligands or the protein will be carried out.
- opt_cart
Optimization of Cartesian atomic coordinates will be done. This is the default optimization type for molecules in vacuum or in solution.
- opt_solid
Optimization of ligand position inside the protein receptor with frozen ligand internal ligand coordinates. Option is not valid and ignored without an input protein. Because it is the simplest and less expensive ligand optimization in the protein, it is the default behavior. The option is therefore only formal and redundant.
- opt_torsions
Optimization of free rotor torsions for molecules in vacuum or torsions and translational/rotational degrees of freedom for ligands inside protein receptors. In the case when the ligand has no rotor torsions, rigid-body optimization will be performed.
- polarH *r*
This option allows partial flexibility of a protein receptor in the optimization of a ligand inside the protein binding side. All polar protein hydrogens distanced by *r* from the ligand will adjust their position upon energy relaxation. Main chain amide NH hydrogens of peptide groups are not included. The molecular potential used for optimization consists of three components:

- (1) Ligand intra-molecular MMFF potential
- (2) Protein-ligand potential
 - a MMFF terms for interaction between polar protein hydrogens and the ligand
 - b Interaction between fixed protein atoms and the ligand
- (3) Protein intra-molecular potential
 - a MMFF terms involving polar protein hydrogens and atoms up to three bonds apart.
 - b Interaction between the rest of the protein and flexible polar hydrogens

The sum of components 2b and 3b might be called “protein-pseudoligand” interaction and is evaluated according to the model selected with the “-protein_elec” option above.

- residue r
Similar to polarH. This option allows for flexibility of complete residues having at least one atom within distance r of any atom of the ligand. There is no upper value of the r parameter, however using a large value will cause even distant residues to be flexible which could lead to poor results. The intention of this option is to allow only residue atoms interacting directly with the ligand to rearrange upon optimization.
- sideC r
Similar to polarH. This option allows for flexibility of all side chains having at least one atom within distance r of any atom of the ligand. Same comment as in the “-residue” above applies with respect to side chains atoms.
- strip_water
Causes removal of water molecules from the input protein when options “-protein” or “-complex” are used.

3.2.4 PVM Options

- pvmconf f.txt
f.txt is a text file which specifies PVM processor configuration. For every host in the cluster it should contain a line:
host host_name n
where n is the number of processors on the host.
- silent
By default the names of the processed molecules are displayed. The use of “-silent” option will suppress this output.

3.3 Output

Every run generates an output molecule file in user selected format (mol2, SD, etc.) containing optimized coordinates for the ligand, a log file pre.log, setup parameter file pre.param and status file pre.status where pre is a default prefix “szybki” unless user defined with an option “-prefix”(see previous Section). The log file provides the following information for each molecule and each conformer:

- Force field parameter set used (MMFF94 or MMFF94s)
- Protein name

- Molecule name and its atom numbers
- Conformer number
- Number of free rotors (in torsional optimization)
- Initial energy
- Initial rms gradient
- Number of optimizer iterations
- Final energy
- Final rms gradient
- RMSD to the initial conformation
- Energy terms of the optimized conformer
 1. Bond-stretch
 2. Angle-bend
 3. Stretch-bend
 4. Out-of-plane bend
 5. Torsion
 6. van-der-Waals
 7. Coulomb
 8. Ligand-Protein Energy
- CPU time

In the case of protein electrostatic option 4 (protein_elec 4) electrostatic energy is broken down into Coulomb, ligand and protein desolvation and solvent screening terms.

An example command line for the torsional optimization of a quinazoline derivative in the active site of p38 kinase with protein-ligand electrostatic model 1, and precalculated PB potential grid is shown below:

```
szybki -ligands 1DI9_lig.SD -protein 1DI9_pro.pdb -protein_elec 1 -opt_torsions -loadPG 1DI9.grd
```

The log file generated by this command line contains the following lines:

```
Units: [energy] = [kcal/mol]
       [gradient] = [kcal/(mol*Å)]
       [RMSD] = [Å]
       [CPU_time] = [s]
```

```
MMFF94
P38 KINASE
```

```
Molecule 4-[3-methylsulfanylanilino]-6,7-dimethoxyquinazoline, 40 atoms
```

Conformer	0
Number of rotors	8
Initial energy	108.33260
Initial rms gradient	178.50875
No of cycles	45
Final energy	70.79504
Final rms gradient	0.27016
RMSD	1.26301
Maximum displacement	2.46396
Energy terms:	
MMFF VdW	29.67447
MMFF Coulomb	-70.43419
MMFF Bond	75.83281
MMFF Bend	32.31980
MMFF StretchBend	8.83243
MMFF Torsion	16.96113
MMFF Improper Torsion	0.11723
Ligand-Protein Energy	-22.50865
__VdW	-22.31871
(Coulomb diel=2.0	-3.90200)
__Solvent screening	-0.18994
CPU time	1.33000

“Ligand-Protein Energy” is the sum of all components contributing to the interaction between the ligand and the protein relevant the protein-ligand electrostatic model selected. Those components are listed separately with the prepending double underscore “__”. Optionally an energy component tabular file can be generated. See above section 3.2.1, options “-tab” and “-tabFile”.

Part III

Toolkit

C++ Examples

First two examples show how to create the `OESzybki` object and use it to minimize an input molecule in vacuum (example 1) and inside the protein (example 2). The next three examples are using the `OEInterface` in order to select different user options.

4.1 Simple optimization of molecule(s) in vacuum

```
1  /*****  
2  Copyright (C) 2006 by OpenEye Scientific Software, Inc.  
3  *****/  
4  #include "openeye.h"  
5  #include "oechem.h"  
6  #include "oeszybki.h"  
7  
8  using namespace OESystem;  
9  using namespace OEChem;  
10 using namespace OESz;  
11  
12 int main(int argc, char* argv[])  
13 {  
14     if(argc!=3)  
15         OThrow.Usage("%s <input> <output>", argv[0]);  
16  
17     oemolistream lig(argv[1]);  
18     oemolostream out(argv[2]);  
19  
20     OEGraphMol mol;  
21     OEReadMolecule(lig, mol);  
22  
23     OESzybki sz;  
24     OESzybkiResults res;  
25     if(!sz(mol, res)) return 1;  
26  
27     OEWriteMolecule(out, mol);  
28     res.Print(OEOUT);  
29     return 0;  
30 }
```

Listing 4.1: Example of simple optimization of molecule(s) in vacuum

The example illustrates also the default behavior of the `OESzybki` object in the case of molecule(s) outside the protein which is the optimization in complete Cartesian space of all atoms.

4.2 Simple minimization of molecule(s) inside the protein

```
1  /*****  
2  Copyright (C) 2006 by OpenEye Scientific Software, Inc.  
3  *****/  
4  #include "openeye.h"  
5  #include "oechem.h"  
6  #include "oeszybki.h"  
7  
8  using namespace OESystem;  
9  using namespace OEChem;  
10 using namespace OESz;  
11  
12 int main(int argc, char* argv[])  
13 {  
14     if(argc!=4)  
15         OThrow.Usage("%s <protein> <ligand> <output>", argv[0]);  
16  
17     oemolistream prt(argv[1]);  
18     oemolistream lig(argv[2]);  
19     oemolostream out(argv[3]);  
20  
21     OEGraphMol mol;  
22     OEReadMolecule(lig, mol);  
23     OEGraphMol protein;  
24     OEReadMolecule(prt, protein);  
25  
26     OESzybki sz;  
27     sz.SetProtein(protein);  
28     OESzybkiResults res;  
29     if(!sz(mol, res)) return 1;  
30  
31     OEWwriteMolecule(out, mol);  
32     res.Print(OEOUT);  
33     return 0;  
34 }
```

Listing 4.2: Example of simple optimization of ligand(s) inside protein

In the above example the input molecule is optimized inside the protein. In this case, because no run type has been determined a default solid body type optimization is performed

4.3 Optimization of a set of molecules in vacuum or solution

In the example 3 (`openeye/examples/example3.cpp`), after command line options are parsed and input/output files opened, the `OESzybki` object is created and optionally set with the Sheffield solvation model, run type, modified VdW potential and a group of fixed atoms:

```
//Szybki object  
OESzybki sz;  
  
//apply solvent model  
if(itf.Get<bool>("-s"))
```

```

if(!sz.SetSolventModel(OESolventModel::Sheffield))
    OEThrow.Warning("Failed to setup Sheffield solvent model");

//select run type
if(itf.Get<bool>("-t"))
    sz.SetRunType(OERunType::TorsionsOpt);
if(itf.Get<bool>("-n"))
    sz.SetRunType(OERunType::SinglePoint);

//remove attractive VdW forces
if(itf.Get<bool>("-a"))
    sz.SetRemoveAttractiveVdWForces(true);

//fix atoms
if(itf.Has<string>("-f"))
    if(!sz.FixAtoms(itf.Get<string>("-f")))
        OEThrow.Warning("Failed to fix atoms for pattern %s",
            itf.Get<string>("-f").c_str());

```

All input multiconformer molecules are then processed in the loop below:

```

//process molecules
OEMol mol;
OETiter<OESzybkiResults> results; //iterator to energy results
while(OEReadMolecule(inpf, mol))
{
    logfile << "\nMolecule " << mol.GetTitle();
    for(results=sz(mol); results; ++results)
        results->Print(logfile);
    outf << mol;
}

```

4.4 Optimization of a set of molecules inside rigid protein receptor

In example 4 (openeye/examples/example4.cpp) optimization of ligands is done inside the protein receptor, so the OESzybki object has to be setup with the input protein:

```
sz.SetProtein(protein);
```

Another feature of the example code is selection of protein-ligand electrostatic model:

```

//select protein-electrostatic model
string emodel = itf.Get<string>("-e");
if(emodel == "VdW")
    sz.SetProteinElectrostaticModel(OEProteinElectrostatics::NoElectrostatics);
else if(emodel == "PB")
    sz.SetProteinElectrostaticModel(OEProteinElectrostatics::GridPB);
else if(emodel == "Coulomb")
    sz.SetProteinElectrostaticModel(OEProteinElectrostatics::GridCoulomb);
else if(emodel == "ExactCoulomb")
    sz.SetProteinElectrostaticModel(OEProteinElectrostatics::ExactCoulomb);

```

4.5 Optimization of a set of molecules inside partially flexible protein receptor

The only significant difference between the previous example 4 and example 5 (openeye/examples/example5.cpp) is selection of partial protein side-chains flexibility:

```
sz.SetSideChainsFlexibility(itf.Get<double>("-d"));
```

OESzybki Classes and Member Functions

5.1 OESzybki

```
class OESzybki
```

Class `OESzybki` provides a high-level API for optimization of molecules in vacuum, solution and in protein-receptor environment using MMFF94/MMFF94s force field. Optimization in solution can be performed using two different solvation models, while the optimization inside the protein can be done with several protein-ligand electrostatic interaction models allowing for limited protein flexibility in the neighborhood of a ligand.

Important feature: The current implementation of the `OESzybki` class does not provide a copy constructor nor an assignment operator, and attempts to use them will give code that will not compile.

5.1.1 Constructor

```
OESzybki(unsigned int force_field=OEForceFieldType::MMFF94)
```

Constructs `OESzybki` object which will use MMFF force field. At the moment MMFF94 and MMFF94s are the only force fields implemented, so passing other values than `OESz::OEForceFieldType::MMFF94` or `OESz::OEForceFieldType::MMFF94s` will cause fatal error with a message: “Not a valid force field”.

5.1.2 operator()

```
OESystem::OEIterBase< OESz::OESzybkiResults > *operator()(OEChem::OEMCMolBase &m)  
bool operator()(OEChem::OEMolBase &m, OESz::OESzybkiResults &res)
```

The first operator performs geometry optimization on all conformers in the passed molecule “m” and updates it with optimized coordinates. Returns an iterator whose elements are `OESzybkiResults` objects. `OESzybkiResults` contains the energy values of the relevant potential terms, total energies and gradients. The complete `OESzybkiResults` structure is given below. The second operator performs the same task on a molecule in a single conformation. A reference to an instance of `OESzybkiResults` structure has to be passed. If

the calculation fails the operator returns false.

5.1.3 ClearFixAtoms

```
void ClearFixAtoms()
```

Removes the fix constraint on any fixed atoms. After the use of this function, positions of all atoms could change upon optimization.

5.1.4 ClearHarmonicConstraints

```
bool ClearHarmonicConstraints()
```

Removes harmonic constrains. Returns “true” if constraints were removed and “false” when the action was not necessary because a constraining potential was not present.

5.1.5 FixAtoms

```
bool FixAtoms(std::string smarts_pattern)
bool FixAtoms(const OESystem::OEUnaryPredicate< OEChem::OEAtomBase > &)
```

Fixes a set of atoms at their initial positions in space. The first function fixes a set of atoms which belong to a SMARTS pattern determined by the parameter “smarts_pattern”. It returns “true” if the SMARTS string passed is valid, “false” otherwise. The second function fixes only those atoms for which the predicate passed is true, and returns “true” if the predicate is valid.

5.1.6 GetCalculateFrozenTerms

```
bool GetCalculateFrozenTerms() const
```

Returns “true” if constant force field terms due to the usage of adaptors are calculated at the end of optimization.

5.1.7 GetCavitySolventParameter

```
double GetCavitySolventParameter() const
```

Returns the cavity solvation factor currently used by OESzybki object.

5.1.8 GetFixPattern

```
const char* GetFixPattern() const
```

Returns the SMARTS string used for fixing atoms.

5.1.9 GetFixPredicate

```
const OESystem::OEUnaryPredicate< OEChem::OEAtomBase >* GetFixPredicate() const
```

Returns a pointer to the predicate used to determine which atoms are fixed.

5.1.10 GetForceFieldType

```
unsigned int GetForceFieldType() const
```

Returns the force field type currently used by OESzybki object. Possible returned values are defined in namespace OESz::OEForceFieldType, described in Section 7.2.

5.1.11 GetHarmonicConstraints

```
const OESystem::OEUnaryPredicate< OEChem::OEAtomBase >*  
GetHarmonicConstraints(double &kc, double &kd) const
```

Provides a method to query for the harmonic constraint used, the force constant “kc”, and distance “kd”. The function returns a pointer to a predicate which determines which atoms are constrained. If the returned pointer is 0, then no harmonic constraints are used.

5.1.12 GetLigandRMSDHeavy

```
bool GetLigandRMSDHeavy() const
```

Returns true if RMSD between the optimized and initial structures are measured using heaving atoms only.

5.1.13 GetOptimizerType

```
unsigned int GetOptimizerType() const
```

Returns the type of currently active optimizer. Possible returned values are defined in the namespace OESz::OEOptType, Section 7.6.

5.1.14 GetPolarHFlexibility

```
double GetPolarHFlexibility() const
```

Returns the distance from the ligand within which protein polar hydrogens are optimized.

5.1.15 GetProtein

```
bool GetProtein(OEChem::OEMolBase &p) const
```

Copies the current protein molecule to the passed molecule object “p”. If the OESzybki object does not hold a protein molecule, the function will return false.

5.1.16 GetProteinDielectric

```
double GetProteinDielectric() const
```

Returns the protein dielectric constant currently used.

5.1.17 GetProteinElectrostaticModel

```
unsigned int GetProteinElectrostaticModel() const
```

Returns the type of protein-ligand electrostatic model currently used by the OESzybki object. Possible values are given in namespace OESz::OEPProteinElectrostatics, Section 7.5.

5.1.18 GetProteinVdWCutoff

```
double GetProteinVdWCutoff() const
```

Returns the value of VdW cutoff range used for the calculation of protein-ligand interaction.

5.1.19 GetRemoveAttractiveVdWForces

```
bool GetRemoveAttractiveVdWForces() const
```

Returns “true” if attractive Van der Waals forces are not present, “false” otherwise.

5.1.20 GetRemoveCoulombTerms

```
bool GetRemoveCoulombTerms() const
```

Returns “true” if Coulomb terms are removed and “false” when they are present.

5.1.21 GetResidueFlexibility

```
double GetResidueFlexibility() const
```

Returns the distance from the ligand within which protein residues are optimized.

5.1.22 GetRunType

```
unsigned int GetRunType() const
```

Returns the run type. Possible returned values are defined in the namespace OESs::OERunType, Section 7.3.

5.1.23 GetSheffieldParameters

```
void GetSheffieldParameters(double &a, double &b, double &dc) const
```

Returns the Sheffield solvation model parameters which are currently used.

5.1.24 GetSideChainsFlexibility

```
double GetSideChainsFlexibility() const
```

Returns the distance from the ligand within which protein side chains are optimized.

5.1.25 GetSolventModel

```
unsigned int GetSolventModel() const
```

Returns the type of solvation model used to optimize molecule(s) in solution. Possible returned values are defined in the namespace `OESz::OESolventModel`, Section 7.4.

5.1.26 GetUseCurrentCharges

```
bool GetUseCurrentCharges() const
```

Returns “true” if the current partial charges are used for solvation energy calculation or for protein-ligand electrostatics. A returned value of “false” indicates usage of MMFF partial charges.

5.1.27 GetVerbose

```
bool GetVerbose() const
```

Returns “true” if verbose mode (see above) is turned on.

5.1.28 LoadPotentialGrid

```
void LoadPotentialGrid(std::string fname)
```

Reads a pre-calculated potential grid from the file name “fname”. For a large number of ligands to be optimized inside a single protein with the use of the electrostatic model `OESz::OEProteinElectrostatics::GridPB` or `OESz::OEProteinElectrostatics::GridCoulomb` (see function `SetProteinElectrostaticModel`), this function offers a significant saving in CPU time.

5.1.29 SavePotentialGrid

```
void SavePotentialGrid(std::string fname)
```

Saves potential grid in the file named “fname”. The potential grid will be generated in protein-ligand calculations in which electrostatic model used is `OESz::OEProteinElectrostatics::GridPB` or `OESz::OEProteinElectrostatics::GridCoulomb` set with the function `SetProteinElectrostaticModel`.

5.1.30 SetCalculateFrozenTerms

```
void SetCalculateFrozenTerms(bool state)
```

Eliminates the calculation of constant force field terms at the end of optimization when the value of the “state” parameter is “false”. This function has no effect for the optimization in full Cartesian coordinates. It allows to save

cpu and memory for partial optimization of large molecules, because by default all potential terms are calculated at the end of optimization.

5.1.31 SetCavitySolventParameter

```
void SetCavitySolventParameter(double area_factor)
```

Sets the cavity solvation factor. If the function is not used and surface solvation is set (see above) a default value of 0.025 will be used.

5.1.32 SetForceFieldType

```
bool SetForceFieldType(unsigned int ff)
```

Returns “true” if the force field selected can be used. The only values currently available for “ff” parameter are defined in the namespace `OESz::OEForceFieldType`, Section 7.2.

5.1.33 SetHarmonicConstraints

```
bool SetHarmonicConstraints(double kc, double kd=0.0,  
const OESystem::OEUnaryPredicate< OEChem::OEAtomBase > &pred=OEChem::OEIsHeavy())
```

Adds potential term $V = k_c(r - k_d)^2$ for every atom for which predicate “pred” is true. Default predicate imposes harmonic potential on every heavy atom. Parameter “kc” is force constant k_c and “kd” is constraining distance k_d in Å. By default k_d is set to 0. Function returns true if harmonic potential added.

5.1.34 SetLigandRMSDHeavy

```
void SetLigandRMSDHeavy(bool state)
```

Replaces the default all atoms RMSD between the initial and optimized structures with heavy atoms RMSD only when the “true” is passed.

5.1.35 SetOptCriteria

```
void SetOptCriteria(unsigned int imax, double gtol=0.0)
```

Sets optimization termination/convergence criteria. If not used, Szybki will use a maximum of 1000 optimization iterations, and a 0.1 threshold for gradient norm $\sqrt{\sum_i g_i g_i}$ to be reached. The default value of 0.0 does not require the complete annihilation of forces, but is used rather as a boolean to tell Szybki to use a 0.1 threshold.

5.1.36 SetOptimizerType

```
void SetOptimizerType(unsigned int type)
```

Sets the optimization method. The available methods are BFGS (default) and Conjugate Gradient. The possible values of a parameter “type” are given in Section 7.6

5.1.37 SetPolarHFlexibility

```
void SetPolarHFlexibility(double dist)
```

Includes all protein polar hydrogens which are within “dist” of the ligand in the set of atoms that can move during optimization. The term ‘polar hydrogens’ means here all hydrogens bonded to non-carbon atoms with the exclusion of main chain amide hydrogens. As in the case of `SetSideChainsFlexibility`, the function should not be abused by adding hydrogens distant from the ligand. In practice it means that only polar hydrogens involved in protein binding should be allowed to change their positions. A reasonable value of “dist” parameter should be set for every system by the user.

5.1.38 SetProtein

```
void SetProtein(const OEChem::OEMolBase &protein)
```

Sets the protein in which the ligand will be optimized. This function has to be used in the case of ligand optimization inside the protein target.

5.1.39 SetProteinDielectric

```
void SetProteinDielectric(double d)
```

Sets the dielectric constant of a protein for the calculation of protein-ligand interaction. When the function is not used, the default value of the protein dielectric constant is 1.

5.1.40 SetProteinElectrostaticModel

```
void SetProteinElectrostaticModel(unsigned int type)
```

Selects the electrostatic model for protein-ligand interaction calculation. The choices for parameter “type” are given in the namespace `OESz::OEProteinElectrostatics`, Section 7.5.

5.1.41 SetProteinVdWCutoff

```
void SetProteinVdWCutoff(double r)
```

Sets the cutoff distance for calculation of protein-ligand VdW interactions. The default value is 18 Å.

5.1.42 SetRemoveAttractiveVdWForces

```
bool SetRemoveAttractiveVdWForces(bool state)
```

Removes attractive Van der Waals forces when the value of parameter “state” is “true” and restores them when value “false” is passed. Successful removal or restoring results in returning “true”, while the returned value of

“false” indicates that the Van der Waals attractive forces are already removed/restored.

5.1.43 SetRemoveCoulombTerms

```
bool SetRemoveCoulombTerms(bool state)
```

Removes Coulomb terms from the force field when the value of parameter “state” is “true” and restores those terms when value “false” is passed. Successful removal/restoring results in returning “true”, while the returned value of “false” indicates that the terms were already removed/restored.

5.1.44 SetResidueFlexibility

```
void SetResidueFlexibility(double dist)
```

Includes those complete residues of the protein which are at least “dist” Å from the ligand, to be optimized along with the ligand. The purpose of this function is to relax only a few residues which are in in close proximity to the ligand. Accordingly it is recommended that large values of “dist” not be used. In most cases no more that 2-3 Å is a good choice.

5.1.45 SetRunType

```
bool SetRunType(unsigned int type)
```

Sets requested run type. Returns “true” if run was successfully set. Possible input choices are given in the namespace `OESz::OERunType`, Section 7.3.

5.1.46 SetSheffieldParameters

```
void SetSheffieldParameters(double a, double b, double dc=1.0)
```

Sheffield solvation energy requires two parameters a and b :

$$E_{solv} = -\frac{f_{\epsilon}}{8\pi\epsilon_0} \sum_{i,j} \frac{q_i q_j}{\sqrt{(aR_i R_j + b r_{ij}^2)}}$$

where q_i and q_j are atomic charges, R_i and R_j are VdW radii for atoms i and j , and r_{ij} is the interatomic distance. Parameter f_{ϵ} depends on dielectric constant of the solute: $f_{\epsilon} = 1/\epsilon - 0.0125$) This function sets both parameters a and b to the required values, as well as the solute dielectric constant (which may not exceed 10). If the function not used, the solute dielectric constant is set to 1.0, and the default values of $a = 1.553149$ and $b = 0.735694$ are adopted (*Grant 2006*).

5.1.47 SetSideChainsFlexibility

```
void SetSideChainsFlexibility(double dist)
```

Includes those side chains of the protein which are at least “dist” Å from the ligand, to be optimized along with the ligand. The purpose of this function is to optimize only a few side chains which are in in close proximity to the ligand. Accordingly it is recommended that large values of “dist” not be used. In most cases no more that 2-3 Å is a good choice.

5.1.48 SetSolventModel

```
bool SetSolventModel(unsigned int type)
```

Sets solvent model if optimization is to be performed on a molecule in solution. Acceptable values or their restricted combinations for parameter “type” are given in the namespace `OESz::OESolventModel`, Section 7.4.

5.1.49 SetUseCurrentCharges

```
void SetUseCurrentCharges(bool state)
```

The quality of a solvation energy depends on the atomic partial charges. By default MMFF charges are used, however another set of partial charges may be used as well. This function prevents the usage of MMFF charges for solvation calculation and in protein-ligand electrostatic model (see next Sections), when the parameter “state” is set to true. Passing “false” will cause returning to MMFF charges.

5.1.50 SetVerbose

```
void SetVerbose(bool verb)
```

Outputs intermediate information from a calculation with the use of the method `OEThrow.Info`.

5.2 OESzybkiResults

```
class OESzybkiResults
```

This class provides access to the results of optimization done with the `OESzybki` object.

5.2.1 Constructors

```
OESzybkiResults()  
OESzybkiResults(const OESzybkiResults &)
```

Constructs `OESzybkiResults` object.

5.2.2 operator=

```
OESzybkiResults &operator=(const OESzybkiResults &)
```

5.2.3 GetCPUTime

```
float GetCPUTime() const
```

Returns CPU time in seconds for the optimization.

5.2.4 GetConfIdx

```
unsigned int GetConfIdx() const
```

Returns conformer id number.

5.2.5 GetEnergyTerm

```
double GetEnergyTerm(unsigned int) const
```

Returns the current energy value for a specified potential term. The integer values which determine potential terms are defined in the namespace `OESz::OEPotentialTerms`, Section 7.1.

5.2.6 GetFinalRMSGradient

```
double GetFinalRMSGradient() const
```

Returns final RMS of forces, $\sqrt{\frac{\mathbf{g} \cdot \mathbf{g}}{n_v}}$, where \mathbf{g} is the gradient vector and n_v the number of variables.

5.2.7 GetFinalTotalPotential

```
double GetFinalTotalPotential() const
```

Returns the value of the optimized potential for the molecular system.

5.2.8 GetInitialRMSGradient

```
double GetInitialRMSGradient() const
```

Returns initial RMS of forces, $\sqrt{\frac{\mathbf{g} \cdot \mathbf{g}}{n_v}}$, where \mathbf{g} is the gradient vector and n_v the number of variables.

5.2.9 GetInitialTotalPotential

```
double GetInitialTotalPotential() const
```

Returns the value of the initial potential for the molecular system to be optimized.

5.2.10 GetInterEnergy

```
double GetInterEnergy() const
```

Returns interaction energy between protein (or DNA) and the ligand optimized inside the macromolecule. For ligands optimized in vacuum or in solution, this function returns 0.

5.2.11 GetMaxDisplacement

```
double GetMaxDisplacement() const
```

Returns the maximum atomic displacement for a single atom in Å during optimization.

5.2.12 GetNumCycles

```
unsigned int GetNumCycles() const
```

Returns the number of cycles performed by the optimizer.

5.2.13 GetNumFixAtoms

```
unsigned int GetNumFixAtoms() const
```

Returns the number of atoms which will be fixed during optimization.

5.2.14 GetNumRotors

```
unsigned int GetNumRotors() const
```

Returns the number of rotatable bonds in the molecule.

5.2.15 GetProteinRMSD

```
double GetProteinRMSD() const
```

In the case of a partially optimized protein (side chains, polar hydrogens in proximity to the ligand) the function returns the RMS displacement of a protein from its initial structure.

5.2.16 GetRMSD

```
double GetRMSD() const
```

Returns RMS displacement of the optimized structure with respect to the initial structure.

5.2.17 GetTotalEnergy

```
double GetTotalEnergy() const
```

Returns the total energy of the optimized system. That includes all intra and inter molecular MMFF terms, solvation energy (Sheffield or PB, if present) and protein-ligand interaction energy if a ligand is optimized inside the protein.

5.2.18 IsActiveTerm

```
bool IsActiveTerm(unsigned int) const
```

Returns “true” if the potential term specified by the parameter “term” is included in the potential function of the system. Possible parameters values are defined in the the namespace `OESz::OEPotentialTerms`, Section 7.1.

5.2.19 Print

```
void Print(OEPlatform::oeostream &) const  
void Print(OESystem::OErrorHandler &) const
```

Both functions allow the generation of log information on the optimized system. The following information is reported:

1. Conformer id
2. Number of fixed atoms (if any)
3. Number of torsions (if any and if optimization in torsion space is true)
4. Initial energy
5. Initial rms gradient
6. Final energy
7. Final rms gradient
8. RMS displacement upon optimization
9. Maximum displacement during optimization
10. Values of potential terms at final geometry

An example of the log generated with the described functions is given below. All energy data are in kcal/mol, gradient data in kcal/(mol*Å).

```
Molecule Methotrexate  
Conformer                0  
Number of rotors         13  
Initial energy           -101.66691  
Initial rms gradient      7.52270  
No of cycles              21  
Final energy             -110.93011  
Final rms gradient        0.05100  
RMSD                     1.21254  
Maximum displacement     2.69919  
Energy terms:  
MMFF VdW                 71.79110  
MMFF Coulomb             -15.46499  
MMFF Bond                21.96457  
MMFF Bend                47.66939  
MMFF StretchBend         2.89955  
MMFF Torsion              16.22202  
MMFF Improper Torsion    1.25630  
Sheffield Solvation      -257.26806
```

OESzybki Functions

6.1 OEGetEnergyTermName

```
const char* OEGetEnergyTermName(unsigned int)
```

Function returns the name of the potential term which corresponds to the integer values listed in the namespace `OESz::OEPoentialTerms`, Section 7.1

6.2 OESzybkiGetRelease

```
const char* OESzybkiGetRelease()
```

Returns current release number of the Szybki toolkit.

6.3 OESzybkiGetVersion

```
unsigned int OESzybkiGetVersion()
```

Returns current version of the Szybki toolkit.

6.4 OESzybkiIsLicensed

```
bool OESzybkiIsLicensed(const char *features = 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 OESzybki's functionality.

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

```
if (!OESzybkiIsLicensed("python"))  
    OETHrow.Warning("OESzybki 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 (OESzybkiIsLicensed(0, expdate))
{
    OThrow.Info("License expires: day: %d month: %d year: %d",
                expdate[0], expdate[1], expdate[2]);
}
```

OESzybki Namespaces

7.1 OEPotentialTerms

In the following namespace the first 7 entries correspond to the total MMFF94 potential terms values (0-6). In the case of partial optimization of the protein, the next 7 entries (7-13) refer to the MMFF terms of the ligand while entries (14-20) are for MMFF terms of the flexible part of the protein. Entries 21 and 22 represent MMFF terms describing VdW and Coulomb interactions between the ligand and the flexible part of the protein. Term `OEPotentialTerms::ProteinLigandInteraction` (23) describes ligand - entire protein interaction according to the protein electrostatic model chosen. The next four entries (24-27) refer to the values of extra terms in the total potential used to optimize a ligand in solution. Those are PB, Sheffield or Cavity solvation energies and the value of the imposed harmonic constraint. Entries 28-34 are used to break down the total protein-ligand interaction energy into components, for example VdW, GridCoulomb etc. The last term (35) represents the interaction of the flexible part of of the optimized protein-ligand system (ligand + flexible part of the protein) with the rest of the protein and as such is not an observable quantity.

```
namespace OEPotentialTerms
{
  const unsigned int MMFFVdW           = 0;
  const unsigned int MMFFCoulomb       = 1;
  const unsigned int MMFFBond         = 2;
  const unsigned int MMFFBend         = 3;
  const unsigned int MMFFStretchBend  = 4;
  const unsigned int MMFFTorsion      = 5;
  const unsigned int MMFFImproperTorsion = 6;
  const unsigned int LigandMMFFVdW    = 7;
  const unsigned int LigandMMFFCoulomb = 8;
  const unsigned int LigandMMFFBond   = 9;
  const unsigned int LigandMMFFBend   = 10;
  const unsigned int LigandMMFFStretchBend = 11;
  const unsigned int LigandMMFFTorsion = 12;
  const unsigned int LigandMMFFImproperTorsion = 13;
  const unsigned int ProteinMMFFVdW   = 14;
  const unsigned int ProteinMMFFCoulomb = 15;
  const unsigned int ProteinMMFFBond  = 16;
  const unsigned int ProteinMMFFBend  = 17;
  const unsigned int ProteinMMFFStretchBend = 18;
  const unsigned int ProteinMMFFTorsion = 19;
  const unsigned int ProteinMMFFImproperTorsion = 20;
  const unsigned int ProteinLigandMMFFVdW = 21;
  const unsigned int ProteinLigandMMFFCoulomb = 22;
  const unsigned int ProteinLigandInteraction = 23;
  const unsigned int SheffieldSolvation = 24;
  const unsigned int HarmonicConstraint = 25;
}
```

```
const unsigned int PBSolvation           = 26;
const unsigned int CavitySolvation       = 27;
const unsigned int VdWProteinLigand      = 28;
const unsigned int CoulombProteinLigand  = 29;
const unsigned int ProteinDesolvation     = 30;
const unsigned int LigandDesolvation      = 31;
const unsigned int SolventScreening      = 32;
const unsigned int GridCoulomb           = 33;
const unsigned int ExactCoulomb          = 34;
const unsigned int ProteinPseudoLigandInteraction = 35;
}
```

7.2 OEForceFieldType

Entries in this namespace refer to the two force field parameter sets implemented in MMFF used by Szybki

```
namespace OEForceFieldType
{
    const unsigned int MMFF94 = 0;
    const unsigned int MMFF94s = 1;
}
```

7.3 OERunType

This namespace defines four types of currently optimized runs. `OERunType::SolidBodyOpt` is a valid choice only when the ligand is being optimized in the field of a protein. Selecting optimization in torsion coordinates for a molecule inside a protein will also include translational and rotational coordinates to the optimization space.

```
namespace OERunType
{
    const unsigned int SinglePoint = 0;
    const unsigned int TorsionsOpt = 1;
    const unsigned int CartesiansOpt = 2;
    const unsigned int SolidBodyOpt = 3;
}
```

7.4 OESolventModel

Parameter names are self-explanatory: “Sheffield” stands for the Sheffield solvation model defined in the equation from section 5.1.38 describing `SetSheffieldParameters` function, “PB” stands for Poisson-Boltzmann model, and “Cavity” for solvation effects due to the formation of a cavity in the solvent by the solute molecule. Combinations are done with the binary `|` operator. One cannot combine Sheffield and Poisson-Boltzmann models, of course. The function returns true when selected solvation terms are successfully added to the overall potential.

```
namespace OESolventModel
{
    const unsigned int None = 0;
```

```
const unsigned int Sheffield = 0x01;
const unsigned int PB       = 0x02;
const unsigned int Cavity   = 0x04;
}
```

7.5 OEProteinElectrostatics

This namespace describes five different protein electrostatic models available in Szybki. They are:

- No electrostatic interaction between protein and the ligand
- Poisson-Boltzmann solvent screening calculated with PB grid potential
- Coulombic protein-ligand interaction calculated with Coulombic grid
- Exact Coulombic protein-ligand interaction
- Exact Coulombic plus solvation PB energy at each optimization step

```
namespace OEProteinElectrostatics
{
    const unsigned int NoElectrostatics = 0;
    const unsigned int GridPB          = 1;
    const unsigned int GridCoulomb     = 2;
    const unsigned int ExactCoulomb    = 3;
    const unsigned int SolventPBForces = 4;
}
```

7.6 OEOptType

This namespace describes two optimization methods available in Szybki: BFGS and Conjugate Gradient method.

```
namespace OEOptType
{
    const unsigned int BFGS = 0;
    const unsigned int CG   = 1;
}
```

Release Notes

8.1 Szybki 1.4.0

8.1.1 Bug fixes

1. A bug causing memory corruption when the function `SetSideChainsFlexibility()` was called after `SetProten()` was fixed.
2. Memory leak in ligand torsion optimization inside the partially flexible protein receptor was removed.

8.1.2 New features

1. Starting from the current release, certain classes of divalent selenium compounds can be handled by Szybki. Currently only some compounds in which selenium is bonded to the hydrogen or carbon atoms can be processed. Those compounds include selenols (RSeH) and selenides (RSeR1) in which Se is bonded to alkyl or aromatic carbon atoms, with the following exceptions: (1) Se makes a bond with alkyl C which belongs to the 3- or 4-membered ring, (2) Se makes a bond with aromatic C which belongs to 5-membered ring. In the restriction above MMFF94 aromaticity model is adopted, so for example a pyridine derivative, pyridine-3-selenol can be processed, but a pyran derivative 2H-pyran-3-selenol cannot.
2. Previous Szybki releases attempted to assign the “closest” MMFF94 atom type to those atoms for which a unique MMFF94 type could not be found. Starting from the 1.3.5 release this feature is removed, so consequently the input molecules containing such atoms will not be processed.
3. Ligand RMSD which might be extracted from `OESzybkiResults` object, can optionally refer to heavy atoms only. A new public function: `void SetLigandRMSDHeavy(bool)` in `OESzybki` class has been added for that purpose.

8.2 Szybki 1.3.4

8.2.1 Bug fixes

1. Input proteins with only partial information about residues were not handled properly. As a result none of the side chains which supposed to be made flexible with the use of an option “-sideC” were optimized. This problem was fixed.
2. A bug causing memory corruption in the case of ligand optimization in partially flexible protein receptor is fixed. This bug was fatal in approximately 5% of such optimizations on 64 bits platforms.
3. Residues with non-standard name “CYX” were not properly handled in the situation when their side chains were included into the flexible part of the protein receptor. This bug has been fixed.
4. Member function ClearFixAtoms() did not cleared properly previously fixed atoms. As a result, in spite of its use some atoms were still fixed. This bug has been fixed.

8.2.2 New features

1. Calculation of the constant potential terms at the end of adapted optimization can be optionally eliminated, by the usage of the function “void SetCalculateFrozenTerms(bool)”. This feature allows to save memory and cpu in the case of partial optimization of large molecules. In Szybki application it is achieved by the usage of the “-neglect_frozen” flag.

8.3 Szybki 1.3.3

8.3.1 Bug fixes

1. Until current release, molecular output files in the .oeb format had MMFF aromaticity. In practice, such a situation lead to the visual change between the input and output aromaticity of some molecules. Starting from now OpenEye aromaticity model is used in the .oeb output files.

8.4 Szybki 1.3.2

8.4.1 Bug fixes

1. A bug which occasionally caused errors in fixing a subset of atoms, was fixed.
2. When using PVM, Szybki now runs a mechanism for checking versions used by master and slaves. This mechanism prevents the situation of different versions of Szybki running on the master and slaves.

3. Molecules containing atoms for which MMFF types are not properly typed are not processed. In previous versions attempts were made to assign for such atom the closest MMFF type, however this procedure almost always lead to errors. This situation occurred for example in the case of bad pdb files. For proteins, better diagnostics (residue name and number, atom name) are included as output.
4. Reading protein data in .oeb binary format containing additional untagged information (FRED receptor files) is allowed. Until now using such a file in pym mode caused a fatal error.

8.5 Szybki 1.3.1

8.5.1 Bug fixes

1. The combination of options “-complex if1” and “-ligands if2” run in the pym mode caused incorrect behavior in the case when the preexisting ligand in the macromolecule (in the file if1) overlapped with one or more ligands from the input file if2.
2. Run time license for the shape toolkit is added. It is required when the combination of options “-complex if1” and “-ligands if2” is used.
3. Redundant warnings generated upon “-protein_elec none” and “-strip_water” were removed.
4. Spelling in the log file was fixed.
5. Year 2008 was added to the Szybki banner.
6. In the documentation description for the “-fix_smarts” option an example was added which illustrates the adjustment of hydrogen atom positions.
7. In section 3.3, the examples of log files generated upon ligand optimization in the partially flexible protein followed by a single point PB calculations are given.

8.6 Szybki 1.3.0

8.6.1 Bug fixes

1. A bug causing crash in the case of combined usage of partial protein flexibility and protein electrostatic model `OEProteinElectrostatics::SolventPBFForces` was fixed.
2. A bug causing different behavior and/or occasional crash depending of the order of calls: `SetProtein(const OEChem::OEMolBase&)`, `SetRunType(unsigned int)` and `SetResidueFlexibility(double)` has been fixed.
3. Modification of the potential function after the usage of coordinate adaptors could occasionally result in crash, for example when function `FixAtoms()` was followed by `SetRemoveCoulombTerms()`. This bug was fixed.

4. Running in the pvm mode requires now setting the PVM_PATH variable, which points at the directory where the Szybki script is located. For example if Szybki was untared in the directory “mydir”, then using bash shell one can set the above variable with:
export PVM_PATH=mydir/openeye/bin

8.6.2 New features

1. Optimization using conjugate gradient method was added as an alternative to the default quasi-Newton method with the BFGS Hessian update scheme. In Szybki toolkit this is done by the usage of a new function “void SetOptimizerType(unsigned int type)” described in subsection 5.1.32. In Szybki application a flag “-conj” has been added for that purpose.
2. Default value of the protein dielectric constant is 1. Previous default value for that parameter was 2, however it was not consistent with the default value of MMFF94 force field.
3. Protein-electrostatic models in Szybki application are selected with descriptive strings. Available options are:
 - “None”(“none”) - no electrostatic potential used (default option)
 - “ExactCoulomb” - exact MMFF Coulomb potential
 - “GridCoulomb” - MMFF Coulomb potential on grid
 - “PB” - PP solvent forces used at every step

8.7 Szybki 1.2.2

8.7.1 Bug fixes

1. In the case when protein electrostatic model was set at “OEProteinElectrostatics::NoElectrostatics” and partial protein flexibility was allowed with the “void SetSideChainsFlexibility(double)” or “void SetPolarHFlexibility(double)” functions, Coulombic terms describing the interaction between the ligand and flexible protein atoms were included during optimization. This was inconsistent with the meaning of “NoElectrostatics”, therefore those Coulombic terms were eliminated in the current release
2. Using the function “void SetSideChainsFlexibility(double)” resulted occasionally in double counting of some residues. This bug was fixed.
3. Fixing atoms with a function “bool FixAtoms(std::string)” where the passed string is SMARTS pattern, was based on MMFF aromaticity types. Starting from the current release OE aromaticity model is used.

8.7.2 New features

1. Two new functions are added to the API:

```
void SetResidueFlexibility(double dist);  
double GetResidueFlexibility() const;
```

These allow the user to introduce and query for the flexibility of entire residues around the ligand.

BIBLIOGRAPHY

- [1] 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.
- [2] 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.
- [3] 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.
- [4] 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.
- [5] 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.
- [6] T.A. Halgren, “MMFF VI. MMFF94s Option for Energy Minimization Studies”, *Journal of Computational Chemistry*, Vol. 20, pp. 720–729, 1999.
- [7] 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.
- [8] J.A. Grant, B.T. Pickup, M.J. Sykes, C.A. Kitchen and A. Nicholls “A Simple Formula for Dielectric Polarization Energies: The Sheffield Solvation Model”, *Chem. Phys. Letters*, Vol. 441, pp. 163–166, 2007.