



OpenEye
Scientific Software

QUACPAC

Release 1.5.0

OpenEye Scientific Software, Inc.

November 08, 2011

CONTENTS

1	Front Matter	1
2	Installation and Platform Notes	3
2.1	Licenses	3
2.2	Installation	3
2.3	Uninstallation	5
3	QUACPAC Theory and Usage	7
3.1	Introduction to QUACPAC	7
3.2	<i>tautomers</i> - Enumeration and Canonicalization	7
3.3	<i>tautomers</i> Usage	7
3.4	<i>pkatyper</i> - Ligand pKa	9
3.5	<i>pkatyper</i> Usage	10
3.6	<i>fixpka</i> - pKa Normalization	11
3.7	<i>fixpka</i> Usage	12
3.8	<i>molcharge</i> - Partial Charges	12
3.9	<i>molcharge</i> Usage	14
4	Release Notes	17
4.1	QUACPAC 1.5.0	17
4.2	QUACPAC 1.3.1	18
4.3	QUACPAC 1.3.0	18
4.4	QUACPAC 1.1.0	18
	Bibliography	21
	Index	23

FRONT MATTER

Copyright 1997-2011 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 Accelrys, 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.

INSTALLATION AND PLATFORM NOTES

2.1 Licenses

To run QUACPAC and the associated utilities you will need to obtain a license file for QUACPAC from OpenEye Scientific Software (business@eyesopen.com). The license file should be in a file pointed to by the **OE_LICENSE** environment variable.

2.2 Installation

2.2.1 General Installation

By default, all OpenEye applications are installed into a single distribution directory tree on the specified machine. The default location for this tree is platform specific and will be detailed below.

The root of the tree (i.e. the `openeye` directory) contains the following subdirectories:

- admin** This directory is intended to contain any administrative scripts and tools associated with the installed applications. Currently, this directory is simply a placeholder on all platforms except for Microsoft Windows, where it contains the uninstaller executables.
- arch** This directory contains the collection of platform specific subdirectories. Each subdirectory contains the actual installed executables and support libraries for the associated platform. In the platform specific subdirectory there will be a subdirectory for each application. Within that will be another subdirectory for each version of that application.
- bin** This directory contains a startup script for each application that has been installed. This script determines, at run-time, what the current platform is and then calls the appropriate executable in the `arch`. This script enables the easy co-existence of multiple platforms and versions of any OpenEye application in the same distribution tree.
- data** This directory contains all of the associated data for the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.
- docs** This directory contains all of the documentation associated with the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.

examples This directory contains all of the examples associated with the installed applications. There will be a subdirectory for each installed application and within that subdirectory there will be another subdirectory for each specific version of that application.

The startup script discussed in the section on the `bin` directory above will have the same name as the installed executable with which it is associated. When the script is called, it will attempt to determine the current platform and run the appropriate executable if installed. If an appropriate executable cannot be found, the script will report that information, as well as a list of the currently installed platforms. The auto-detection can be overridden by setting one of two environment variables:

- **OE_ARCH** can be used to specify a colon separated list of compatible distributions for the current platform such as:

```
redhat-RHEL5-x64:redhat-RHEL4-x64
```

Specification of this environment variable overrides the auto-detection process, if it is present. If none of the compatible distributions listed are found, the script will fall back to the auto-detection process.

- **APPNAME_OE_ARCH** can be used to specify a colon separated list of compatible distributions for a specific application (as specified by changing the **APPNAME** text in the environment variable name) just like **OE_ARCH** as detailed above.

Specification of this environment variable overrides the **OE_ARCH** environment variable as well as the auto-detection process. If none of the compatible distributions listed are found, the script will fall back to the **OE_ARCH** list first and then to the auto-detection process.

Specifying this variable provides a simple way to customize the behavior for individual applications on non-standard platforms.

The startup script also supports a few commandline arguments including:

- | | |
|---------------------|--|
| -path | Specifying this argument will output the full path of the executable to be run. The executable will not be started if this argument is present. |
| -print_arch | Specifying this argument will output the details of the current platform as detected by the script as well as which platform-version of the executable is being run. The executable will be started if this argument is present. |
| -use_version | Specifying this argument followed by a specific version number allows the user to control which released version of the executable to run. |

2.2.2 Linux/Unix

Linux/Unix distributions are provided as a gzipped tarball of the distribution tree described above. Installation is performed by untarring the file in the desired location. Multiple distributions can be installed in the same location without any challenge.

To ensure that the installed applications can be called from the command line, be sure to add the full path of the `openeye/bin` subdirectory to the **PATH** environment variable. For instance, if the distribution was installed into `/usr/local/openeye`, the **PATH** environment variable should contain: `/usr/local/openeye/bin`.

2.2.3 Windows

Windows distributions are provided as a standard EXE installer. By default, all OpenEye applications will install into the `C:\OpenEye` directory.

An OpenEye group with an application specific subgroup will be added to the *Start* menu. The application specific subgroup will contain links to the documentation, the uninstaller, as well as to a Windows command shell which has

the appropriate **PATH** settings already defined to allow the user to simply type the executable name at the prompt without concern for where the executable is actually installed.

For GUI applications, a link to the application will be created on the desktop as well as in the application specific subgroup of the *Start* menu.

2.2.4 Mac OS X

Mac OS X distributions are provided as a standard *pkg* installer delivered as a *dmg* disk image. By default, all OpenEye applications will install into the `/Applications/OpenEye` directory.

To ensure that the installed applications can be called from the command line in the *Terminal*, be sure to add `/Applications/OpenEye/bin` to the **PATH** environment variable.

For GUI applications, an application bundle which can be clicked on to start, will be present in the `/Applications/OpenEye` directory. This bundle cannot be moved independent of the `OpenEye` directory. For instance, the entire `OpenEye` directory can be moved as one piece, but moving the application bundle or the contents of any of the subdirectories in the `OpenEye` directory may cause the application to not start. However, the bundle can still be dragged into the Dock and run from there without any problem.

2.3 Uninstallation

2.3.1 Linux/Unix

To uninstall a single distribution of a product the relevant subdirectories for that product and version simply need to be deleted from within the following directories:

arch In the `openeye/arch` directory is a platform specific subdirectory. Within this are directories for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled. For example, to delete or uninstall v1.0.0 of a product, delete the folder “<product_name>/1.0.0”.

data In the `openeye/data` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

docs In the `openeye/docs` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

examples In the `openeye/examples` directory is a subdirectory for each installed product and within those are subdirectories for each version of the product. Delete the subdirectory for the version which is to be uninstalled.

2.3.2 Windows

Installation of an OpenEye product on Windows causes an OpenEye group with an application specific subgroup to be added to the *Start* menu. One of the items in the application specific subgroup is a link to the uninstaller. Clicking on the uninstaller initiates a wizard which guides the user through uninstallation.

For GUI applications, uninstallation also removes the desktop link to the application as well as in the application specific subgroup of the *Start* menu.

2.3.3 Mac OS X

To uninstall a single distribution of a GUI application simply drag the application from /Applications/OpenEye/bin to the Trash can.

To uninstall a single distribution of a command line application you will need to delete the executable/folder from /Applications/OpenEye/arch/osx-10.6-x64/. For example, to delete or uninstall v1.0.0 of a product, delete the folder “1.0.0” located in /Applications/OpenEye/arch/osx-10.6-x64/<product_name>.

Associated documentation, data and example files for a single distribution can be uninstalled by deleting the subdirectory/folder from within the /Applications/OpenEye/data, /Applications/OpenEye/docs and /Applications/OpenEye/examples directories.

QUACPAC THEORY AND USAGE

3.1 Introduction to QUACPAC

The chemistry of molecular interactions is a matter of shape and electrostatics, but doing electrostatics poorly is worse than doing none at all; accurate charges are required. Even the best charge models are useless if protonation states are wrong. QUACPAC attempts to offer everything necessary to do charges correctly. It includes pKa and tautomer enumeration in order to get correct protonation states, partial charges using multiple models that cover a range of speed and accuracy, and electrostatic potential map construction and storage.

3.2 *tautomers* - Enumeration and Canonicalization

OpenEye Scientific Software's *tautomers* program is used for canonicalizing and/or enumerating the tautomeric forms of a small molecule. Canonicalization converts any of the tautomeric forms of a given molecule into a single unique representation. This is useful for database registration where alternate representations of tautomeric compounds often leads to duplicate entries in a database.

Some effort is made by the *tautomers* program to direct the "canonical" representation to be a physiologically preferred form. However, there are no guarantees the tautomer selected is indeed the lowest energy and, indeed, solvent effects, etc., preclude there being a single "best" form of a tautomer. Fortunately, this is not necessary for database work.

tautomers is not a conformer generation program and will not create coordinates for molecules that are read in with no coordinates. When used on molecules with three-dimensional coordinates, *tautomers* attempts to place hydrogens in a reasonable manner. However, *tautomers* does not modify the heavy-atom coordinates of the molecule. In cases where the change in tautomer-state dictates a change in conformation, one will need to use a conformer-generation tool (such as OMEGA) to generate reasonable conformations for the output from *tautomers*. We recommend that in the preparation of small-molecules for study, charge-state and tautomer enumeration be performed before conformer generation.

3.3 *tautomers* Usage

3.3.1 Required Parameters

-in

File containing one or more molecules for which tautomers will be generated. An input file is required, but the *-in* flag is optional. The first parameter listed with no flag will be automatically mapped to the input file. Unflagged parameters must occur last in the parameter list. Any OpenEye supported molecule format can be used for input.

Optional parameters

-all

Generate all tautomers, including high-energy tautomers and methyl-group tautomerism. By default, *tautomers* enumerates just the lowest energy class of tautomers for each input structure. This option is equivalent to the command line options *-level 7* and *-ch3*. [default = false]

-can

Output the OpenEye canonical SMILES for each tautomer. By default, *tautomers* writes arbitrary SMILES where the order of the atoms in each tautomer is the same for a given input molecule, which often makes it easier to see the differences between tautomers when reading the SMILES. [default = false]

-ch3

Allows carbon atoms that are close by appropriate heteroatoms to change hybridization state. This permits structures such as cyclohexa-2,4-dien-1-one to be considered a tautomer of phenol, and other examples of keto-enol tautomerism. This option is implied by the *-all* command line option. Unfortunately, the *-ch3* flag may cause the calculation time to increase by many orders of magnitude for some molecules. [default = false]

-count

Output just the number of tautomers per compound rather than listing the tautomers. [default = false]

-kekule

Output is in Kekule format for *.smi*, *.ism*, and *.can*. Disabling aromaticity often makes it easier to understand the differences between tautomers. [default = false]

-level

Manually set the acceptable approximate tautomer energy level to use in enumeration. This takes an integer value between zero and seven inclusive, where zero corresponds to lowest energy states and seven corresponds to the highest energy state. The *-all* command line option increases this value to seven. By default, the *tautomers* program enumerates all tautomers in the lowest non-empty low energy state; first trying level zero and if no tautomers are found increasing to one, then two and so on. [default = 0]

-max

Specify a maximum number of tautomers to enumerate for a single input structure. Over 99% of compounds require less than 100 tautomers, indeed most organic compounds gave only a single tautomer, however some pathological dyes and chromophores may individually have nearly a million possible tautomeric forms. The current default is a limit of 1000 tautomers per input structure. [default = 1000]

-out

Output filename, where the file extension indicates the output format. All OpenEye supported molecule formats are allowed but *.smi* and *.ism* are recommended. The default setting will print *.smi* format to standard out.

-param

The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. *tautomers* generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by *tautomers* is created by combining the prefix base name with the *.parm* extension.

-paramfile

The filename for the output parameter file can be set with this flag.

-prefix

Similar to *-paramfile*, the parameter file will be written to what follows this flag plus the extension *.param*. [default = *tautomers*]

-reasonable

Only for those who want a unique tautomer which is reasonable looking. The program will output the most

aromatic tautomer of the first 64 attempted. [default = false]

-savestereo

This allows for any atom or bond with stereochemistry set to not take part in tautomerization. Without this setting it is possible for stereochemistry to be removed during tautomerization. [default = false]

-uniq

Run in canonicalization mode, where for each molecule in the input file, a single “canonical” tautomer is written to the output file. By default, *tautomers* runs in enumeration mode. [default = false]

3.3.2 Example Executions

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

Consider the following input file, *guanine.smi*, that contains just the following line: c1[nH]c2c(=O)[nH]c(nc2n1)N

```
prompt> tautomers guanine.smi output.smi
```

Which should write the following 15 structures to the file *output.smi*.

```
c1[nH]c2c(=O)nc([nH]c2n1)N
c1[nH]c2c(=O)[nH]c(nc2n1)N
c1nc2c(=O)nc([nH]c2[nH]1)N
c1nc2c(=O)[nH]c(nc2[nH]1)N
c1[nH]c2c(=O)[nH]c(=N)[nH]c2n1
c1nc2c(=O)[nH]c(=N)[nH]c2[nH]1
c1[nH]c2c(nc(nc2n1)N)O
c1nc-2c(nc([nH]c2n1)N)O
c1nc2c(nc(nc2[nH]1)N)O
c1nc-2c([nH]c(nc2n1)N)O
c1[nH]c2c(nc(=N)[nH]c2n1)O
c1[nH]c2c([nH]c(=N)nc2n1)O
c1nc2c(nc(=N)[nH]c2[nH]1)O
c1nc-2c([nH]c(=N)[nH]c2n1)O
c1nc2c([nH]c(=N)nc2[nH]1)O
```

A more exhaustive enumeration of the tautomers of guanine can be performed using the *-all* (or *-a*) option.

```
prompt> tautomers -all guanine.smi output_all.smi
```

which will write a total of 1000 guanine tautomers to the file *output.smi*. This can be verified using the *-count* command line option, sending the output to the screen.

```
prompt> tautomers -all -count guanine.smi output_all.smi
```

```
1000
```

3.4 *pkatyper* - Ligand pKa

3.4.1 Introduction

Assessment of ligand pKas can be broken into two phases. The first phase is enumeration of the protonation states of interest, and the second phase is assigning a pKa value to each of these states. An intermediate phase of assigning

microscopic pKas to each of the atomic-deprotonations may also be considered.

It is common in the course of modeling small-molecules to explore the conformational ensemble of the small molecule. Often structures as high as 5-8 kcal/mol above the aqueous ground-state can be important to biological processes. It is also appropriate to enumerate a protonation-state ensemble of the small molecule.

Similar to *tautomers*, OpenEye has a solution for enumerating reasonable protonation states, but not for assessing the energetics of the state (e.g. assigning a pKa value). OpenEye's solution for pKa enumeration seeks to enumerate all of the pKa states that fall roughly in the pH range of 2-14 in aqueous solvent. This range of pKa values generates an ensemble that includes the ground-state plus all charge states within 8 kcal/mol ΔG . This value was chosen to correspond to the similar range that is often used for generating conformational ensembles of small molecules.

3.4.2 Theory

pkatyper enumerates charge states based on primary, secondary and tertiary atom types of each atom in a molecule. The primary atom type is based on the atom's group and its valence. The primary atom-type defines the atom's basic propensity to support a formal charge. The secondary atom-type is defined by the atom-type of the neighbors for each atom. These secondary atom-types, such as aromaticity, alpha-beta unsaturation, or electronegative-groups, modulate each atom's basic propensity to support formal charges. The tertiary atom-types assess the effects of nearby formal charges on a given atom's formal charge. The combination of the primary, secondary and tertiary atom-types determine which formal charge states are allowed for each atom in a molecule. The primary and secondary atom-types are determined once, while the tertiary atom-types are determined as part of the enumeration process.

pkatyper is a rudimentary approach to pKa prediction. While *pkatyper* is not suited for prediction of absolute pKas, it is quite amenable to enumeration of all reasonable charge states of a very wide variety of small-molecule chemistries.

pkatyper is not a conformer generation program and will not create coordinates for molecules that are read in without coordinate. When used on molecules with three-dimensional coordinates, *pkatyper* attempts to place new hydrogens in a reasonable manner. However, *pkatyper* does not modify the heavy-atom coordinates of the molecule. In cases where the change in protonation-state dictates a change in conformation, one will need to use a conformer-generation tool (such as OMEGA) to generate reasonable conformations for the output from *pkatyper*. We recommend that in preparation of small-molecules for study, charge-state and tautomer enumeration be performed before conformer generation.

In the future, OpenEye will release a product which assigns a pKa value to each of the enumerated states.

3.5 *pkatyper* Usage

3.5.1 Required Parameters

-in

Input file of molecules. An input file is required, but the *-in* flag is optional. The first parameter listed with no flag will be automatically mapped to the input file. Unflagged parameters must occur last in the parameter list.

Optional parameters

-count

Only count the number of pKa states rather than enumerating each of the states. If the *-count* flag is specified true, then the output file will contain the name of each compound followed by the number states of that molecule. [default = false]

-max

This integer parameter is the maximum number of pKa states which will be enumerated for any single molecule. If a value of zero is passed to this parameter, no limit will be set. [default = 100]

-out

Output file of molecules. Both the output file and the *-out* flag are optional. The second parameter listed with no flag will be automatically mapped to the output file. Unflagged parameters must occur last in the parameter list. If no output is specified at all, output will be written to *std::out* in SMILES format. A description of supported file formats can be found in the final chapter of this document. If the *-count* flag is specified true, then molecular format is no longer relevant to the output file.

-param

The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. *pkatyper* generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by *pkatyper* is created by combining the prefix base name with the *.parm* extension.

-paramfile

The filename for the output parameter file can be set with this flag.

-prefix

Similar to *-paramfile*, the parameter file will be written to what follows this flag plus the extension *.param*. [default = *pkatyper*]

3.5.2 Example Commands

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

```
prompt> pkatyper drugs.sdf enumerated_drugs.smi
```

Reads the file *drugs.sdf* in SD format, enumerates the pKa states of each molecule and writes them into the file *enumerated_drugs.smi*. Molecules with only one identified pKa state are passed through to the output file.

```
prompt> pkatyper -in drugs.sdf -out enumerated_drugs.smi
```

This command generates the exact same behavior as described above.

```
prompt> pkatyper -count drugs.sdf drugCounts
```

This command reads each of the molecules in *drugs.sdf* and counts the number of pKa states. For each molecule, a single line is added to the *drugCounts* output file that contains the molecule title and the number of states.

3.6 *fixpka* - pKa Normalization

fixpka has a rule-based system to set the ionization state of input molecules. If pKa normalization is turned on, the molecule is set to its most energetically favorable ionization state for $pH=7.4$. The rule-based nature of this calculation allows it to be very fast. Further, despite being rule-based, this approach takes into account many secondary charge interactions.

While more advanced levels of theory can be found for predicting ionization states, this method is very well suited to virtual-screening database preparation. However, *fixpka* may not be appropriate for hit-to-lead or lead optimization.

3.7 *fixpka* Usage

3.7.1 Required Parameters

-in

File containing one or more molecules from which pKa normalization will be done. An input file is required, but the *-in* flag is optional. The first parameter listed with no flag will be automatically mapped to the input file. Unflagged parameters must occur last in the parameter list. Any OpenEye supported molecule format can be used for input.

-out

File for writing output. An output file is required, but the *-out* flag is optional. The second parameter listed with no flag will be automatically mapped to the input file. Unflagged parameters must occur last in the parameter list. Any OpenEye supported molecule format can be used for output but *.smi* or *.ism* is recommended.

3.7.2 Example Commands

The following section shows a common command-line execution of *fixpka*.

```
prompt> fixpka drugs.sdf fixpka_drugs.smi
```

Reads the file *drugs.sdf* in SD format, sets the pKa state of each molecule and writes them into the file *fixpka_drugs.smi*.

3.8 *molcharge* - Partial Charges

3.8.1 Introduction

The assignment of appropriate atomic partial charges, both to small molecule ligands and to biopolymers (such as proteins and nucleic acids) is essential to getting meaningful results from any electrostatics calculation.

A molecule may be considered a collection of atomic nuclei and the electrons that surround them. The number of protons in each nucleus defines its atomic number/element. If the number of electrons exactly matches the number of protons in these nuclei, the molecule is neutral and has no net charge. If there are more electrons than protons, the molecule has a net negative charge, and if there are less, the molecule has a net positive charge.

It is both the atomic nuclei and the net charge that define the identity of a molecule. Indeed, this is a representation common to quantum chemistry. Adding or removing electrons (or atoms) from a molecule produces a different molecule.

In the discrete world of cheminformatics, valence bond theory allows the electrons present in a system to be represented in terms of bonds with formal bond orders, and formal charges assigned to particular atoms. The sum of the formal charges is equal to the net charge on the molecule, but which atoms are assigned which formal charges is to some extent arbitrary, i.e., the same molecule may be represented by similar connection tables, but with formal charges assigned to different sets of atoms.

For example, guanidinium may be expressed as either N[C+](N)N with the formal charge assigned to the carbon, or as [NH2+]=C(N)N with the formal charge assigned to an arbitrarily to one of the otherwise equivalent nitrogens. A similar example is a thiocarboxylate group, where either C(=O)[S-] or C(=S)[O-] are both equally appropriate representations of the same chemical functionality.

A zwitterion is an electrically neutral molecule that is represented as containing atoms with positive formal charge as well as atoms with negative formal charge.

Perhaps the most important fact to appreciate when considering formal charges is that they are all a myth. A figment of a chemist's fevered imagination. Valence bond theory is an exceptionally useful and powerful discretized model of the universe. But as with any model of reality, it has its limitations. Formal charges, for all their numerous benefits to mankind, unfortunately, do not exist.

The limitations of describing formal charges with valence bond theory is apparent even within chemoinformatics. Sydnones, for example, are a class of heterocyclic compound that cannot be written using normal covalent bonds without introducing and arbitrarily assigning both positive and negative charges. Similarly, in inorganic chemistry, the ditechneium cation, Te_2^{+5} , causes similar problems where the +5 formal charge cannot be assigned to both technetium atoms without breaking symmetry.

A better model, or approximation, of the wave function describing the distribution of electron density around a molecule is the use of atomic partial charges. A partial charge is a floating point value assigned to each atomic center intended to model the distribution of electrons over a molecule.

Atomic partial charge is yet another approximation, much like the formal charges described above. However, partial charges provide a much better model to describe the electric field, dipole moment and other observable properties of a molecule.

A common limitation of the use of partial charges is the assumption that they are conformationally invariant. Unfortunately, the distribution of electrons around a molecule depends upon the spatial configuration of its nuclei. Some partial charge assignment algorithms, such as the method of Goddard and Rappé, consider these conformational effects, whilst others that are based on quantum mechanics, such as the RESP and AM1BCC methods of Bayly et al., go to great lengths to eliminate conformational effects, for example, by restraining and symmetrizing symmetric atom positions.

3.8.2 Theory

Marsili-Gasteiger Partial Charges

Marsili-Gasteiger partial charges are assigned using a two stage algorithm. In the first stage, seed charges are assigned to each atom in the molecule. For example, carboxylate oxygens are each assigned the value -0.5. During the second stage, these initial charges are then shared across bonds, moving a certain amount of charge from one atom to another. The partial charge moved and its direction is determined by difference in electronegativities of the atoms on each end of the bond. The relaxation algorithm is then iterated several times (by default eight passes), attenuating the charge moved with each iteration. OpenEye does not recommend use of this charge model. However, it is included for comparison.

MMFF94 Partial Charges

The partial charges used by the MMFF94 and MMFF94s force fields are assigned using a four stage algorithm. In the first stage, each atom of the molecule is assigned an MMFF94 atom type. In the second stage, an initial seed partial charge is assigned to each atom based upon its atom type. For a few atom types, the initial partial charge also depends upon the local environment. In the third stage, the initial charges assigned to aromatic rings are shared between all atoms of the aromatic ring. Finally, in the fourth stage, a table of bond charge increments (BCI) is used to move charges across bonds based upon the bond type of the bond (single, double, triple) and the atom types of the atoms at each end.

AM1 Charges

AM1 charges are a set of Mulliken-type charges derived from a semi-empirical quantum-mechanical calculation. For further discussion of this method, please see Dewar et. al.

AM1BCC Charges

AM1BCC charges start with partial charges derived from the AM1 wave-function. In a second stage, bond-charge corrections (BCC) are applied to the partial charges on each atom to generate the final partial charges. For further discussion, please see the work of Chris Bayly.

The method of assigning AM1BCC charges to a set conformations was proposed by Chris Bayly and colleagues. It is based on the following procedure: Coulomb electrostatic energy is calculated for every conformer using the absolute value of MMFF94 partial charges (original negative charges are replaced with their absolute values). The standard AM1BCC calculation is then performed for the lowest electrostatic energy conformer determined in the previous step, and the AM1BCC charges obtained are assigned to all conformers.

OpenEye considers AM1BCC charges to be the best partial charge model currently available.

AmberFF94 Partial Charges

The partial charges used by the AmberFF94 force field are based on fitting quantum mechanical electrostatic potentials (esp). They were developed to address two key issues with earlier esp-fit charge sets: unrealistically high charges on charge centers and the variation of atomic charges with conformation. While the latter should have some basis in electronic structure, numerical instability in the charge fitting process was the source of both these pathologies. AmberFF94 charge sets use restrained esp-fitting (RESP) to control the numerical instabilities and simultaneous multi-conformer fitting to lead to conformation-independent charges that are restricted to individual residues. Particular attention was given to ensure that backbone amides have consistent charges.

3.9 *molcharge* Usage

molcharge has several flags that are used to determine which type of partial charges to utilize. In addition, there is a single flag to control whether structural optimization should be carried out for models which use an AM1 calculation.

With default parameters, *molcharge* will sprout explicit hydrogens and assign MMFF94 partial charges.

3.9.1 Required Parameters

-in

This is the input file. This file can contain molecules in a wide-variety of molecular formats. Some of the partial charging models, such as AM1 and AM1BCC require coordinates in order to calculate charges. While the input file is required, the *-in* flag is optional. If no *-in* flag is specified, the first unflagged parameter is used as the input file.

-out

This is the output file and it is a required parameter. Since the object of *molcharge* is to generate partial charges, it will only write to formats that can specify a partial charge. These formats currently include only *.mol2*, *.mol2H* and *.oeb*. While the output file is required, the *-out* flag is optional. If no *-out* flag is specified, the second unflagged parameter is used as the output file.

Optional parameters

-altlocs

Retain alternate locations/conformations in *.pdb* format input files, rather than dropping all but the first alternate location. This option allows dictionary-based charges such as AmberFF94 to be applied to all atoms in a disordered molecule but should not be used with conformation-dependent charges such as AM1BCC. [default = false]

- am1**
Assign AM1 partial charges to each atom. [default = false]
- am1bcc**
Assign AM1BCC partial charges to each atom. [default = false]
- amberff94**
Assign AmberFF94 partial charges to recognized protein atoms. Atoms that don't have residue and atom names matching a dictionary entry are assigned the value zero. [default = false]
- clear, -none**
Set the partial charges to zero. [default = false]
- formal**
Set the partial charges of each atom to it's formal charge. Unlike the *-none* command line option, this at least preserves the net charge on the molecule. [default = false]
- gasteiger**
Assign Marsili-Gasteiger partial charges to each atom. [default = false]
- initial**
Set the charges to the MMFF94 initial fractional charges. [default = false]
- mmff**
Assign MMFF94 partial charges to each atom. [default = false]
- noh**
Specified a united-atom charge model. First, charges are calculated with explicit hydrogens. Then, each explicit hydrogen is converted to an implicit hydrogen and it's partial charge is added to the partial charge of it's parent heavy-atom. [default = false]
- opls**
Assign OPLS partial charges to recognized protein atoms. Atoms that don't have a residue name and atom name matching a dictionary entry are assigned the value zero. This method is only appropriate for proteins and peptides. [default = false]
- param**
The argument for this flag is the name of a file containing control parameters. The control parameter file acts to either replace or augment the command line interface. All parameters necessary for program execution may be provided in the control parameter file, although any command given explicitly on the command line will supersede options found in the parameter file. *molcharge* generates a new parameter file containing the full set of execution parameters upon every execution. The name of the parameter file written by *molcharge* is created by combining the prefix base name with the *.parm* extension.
- paramfile**
The filename for the output parameter file can be set with this flag.
- prefix**
Similar to *-paramfile*, the parameter file will be written to what follows this flag plus the extension *.param*. [default = molcharge]
- singlePoint**
Do not optimize the coordinates under the AM1 potential before calculating the charges. [default = false]

3.9.2 Example Commonds

An example run of the *molcharge* program is given below.

```
prompt> molcharge drugs.sdf drugs.mol2
```

This executes *molcharge* with the default parameters, to sprout explicit hydrogens and assign MMFF94 partial charges. The file *drugs.sdf* is opened in SD format for input, and the output is written to the file *drugs.mol2* in Sybyl .mol2 format.

RELEASE NOTES

4.1 QUACPAC 1.5.0

4.1.1 New Features

- *fixpka* has been added. This application may be used to set a molecule to an energetically favorable ionization state for pH=7.4. This is the same pH model that was available in the Filter application. Additionally, the perception of acceptable valence states has been improved to include phosphorus as well as aromatic oxygen and sulfur with +1 formal charge.
- In *tautomers*: Major improvement for carbon hybridization. Now carbon atoms with bonds to one, two, or three heavy atoms are able to change hybridization state. However, only carbons close to an appropriate heteroatom are allowed to change hybridization state. Additionally, unreasonable charge states of carbon have been removed. Now the enabling the *-ch3* option will only generate tautomers appropriate for the given level.
- In *tautomers*: Improvement of the *-reasonable* option, particularly involving exocyclic heteroatoms.
- In *tautomers*: Added stereo preservation flag *-savestereo* with a default value of *false*. Stereo chemistry can be lost during creation and removal of double bonds, but if the user desires a certain stereo setting to be preserved this flag will prevent the associated atoms and bonds from taking part in tautomerization.
- *pkatyper* now implicitly uses the *-reasonable true* setting for its internal call to *tautomers*. This provides a better reference tautomer when enumerating pKa states.
- In *pkatyper*: Improvements to pka states.
- In *molcharge*: Aromaticity settings on molecules are now unchanged on returned molecules. Aromaticity may be temporarily changed inside the function while charges are being calculated, but the molecule will have the same aromaticity after the function as before.
- In *molcharge*: AmberFF94 charges can now be applied to standard protein residues.
- In *molcharge*: A new *-altlocs* flag will retain all alternative locations in a disordered .pdb structure when charging. This option allows dictionary-based charges such as AmberFF94 to be applied to all atoms in a disordered molecule (but should not be used with conformation-dependent charges such as AM1BCC).

4.1.2 Bug Fixes

- In *tautomers*, atom-ordering could in rare instances be inconsistent for output molecules. This led to cases where even though all associated tautomers led to the same unique tautomer, the unique tautomer could have different SMILES representations depending on the input structure. This has been fixed.

4.2 QUACPAC 1.3.1

4.2.1 New features

- The distribution and installation of QUACPAC 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 Application Installation section for details.

4.2.2 Bug fixes

- The previous version had a license failure when AM1 charges were calculated. This bug has been fixed.

4.3 QUACPAC 1.3.0

This release represents the first major reworking of the QUACPAC toolkits and applications in several years. Any users of prior versions of QUACPAC will quickly notice several significant changes. First, this version of QUACPAC does not contain a new release of *protein_pka* program. We are in the midst of a major rewrite of the *protein_pka* program. We hope that this work will bring major improvements in the usability, science and analysis of the *protein_pka* program, yet we do not want to delay the release of the entire QUACPAC package waiting for the *protein_pka* program. The current QUACPAC release does not contain *protein_pka*, but it will be included in a future release when the rewrite is complete. In the interim, we hope you find the bug fixes and new features included in this release useful.

The preps and qpd programs will no longer be supported future versions of QUACPAC.

4.3.1 New features

- A reasonable looking tautomer may be calculated by using the *-reasonable* flag.
- A multiconformer method has been added for calculating AM1BCC charges

4.3.2 Bug fixes

- SMILES map indexes and names are no longer lost when outputting molecules.

4.4 QUACPAC 1.1.0

- Special Note: Version 1.4 of *molcharge* and version 1.2b of the library have been released and inserted into version 1.1 of QUACPAC. Both of these have removed support for the VC2003 partial charging method.
- Version 1.1 is the first full stable release of QUACPAC. QUACPAC remains a heterogeneous release including five primary applications and a programming library with a C++ api.
- *tautomers* is in version 2.0. It provides enumeration of energetically reasonable tautomers of input molecules.
- *pkatyper* is in version 1.1. *pkatyper* provides enumeration of protonation states (pKa ~2-~14).
- *molcharge* is in version 1.3. *molcharge* provides MMFF, AM1, AM1BCC, VC2003 and other partial charges on small molecules.

- *protein_pka* is in version 1.3. *protein_pka* carries out PB calculations to assess the shifts in protein residue pKa's.
- This release includes the oeproton library. The oeproton library exposes the features of *pkatyper*, *tautomers* and *molcharge* in three high level C++ api points. The version of the library in this release is 1.1b. However, the beta moniker signifies only that at this time, OpenEye reserves the right to modify this api. We believe the quality of the code in this library is very solid and the beta designation does not reflect its quality.

BIBLIOGRAPHY

- [Alexov-1999] E. Alexov and M.R. Gunner, **Incorporating Protein Conformational Flexibility into pH-titration Calculations: Results on T4 Lysozyme**, *Biophysical Journal*, Vol. 74, pp. 2075-2093, **1999**.
- [Baker-1934] John W. Baker, *Tautomerism*, D. Van Nostrand Company Inc. Publishers, **1934**.
- [Dewar-1985] M.J.S Dewar, E.G. Zoebisch, E.F. Healy and J.J.P. Stewart, **AM1: A New General Purpose Quantum Mechanical Model**, *Journal of the American Chemical Society*, Vol. 107, pp. 3902-3909, **1985**.
- [Elguero-1976] Jose Elguero, Claude Marzin, Alan R. Katritzky and Paolo Linda, **The Tautomerism of Heterocycles**, Supplement 1, *Advances in Heterocyclic Chemistry*, Academic Press, **1976**.
- [Ertl-1997] Peter Ertl, **Simple Quantum Chemical Parameters as an Alternative to the Hammett Sigma Constants in QSAR Studies**, *Quantitative Structure-Activity Relationships (QSAR)*, Vol. 16, pp. 377-382, **1997**.
- [Gasteiger-1978] J. Gasteiger and M. Marsili, **A New Model for Calculating Atomic Charges in Molecules**, *Tetrahedron Letters*, pp. 3181-3184, **1978**.
- [Gasteiger-1980] J. Gasteiger and M. Marsili, **Iterative Partial Equalization of Orbital Electronegativity - A Rapid Access to Atomic Charges**, *Tetrahedron*, Vol. 36, pp. 3219-3228, **1980**.
- [Talgren-1996] 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**.
- [Jakalian-2000] Araz Jakalian, Bruce L. Bush, David B. Jack and Christopher I. Bayly, **Fast, Efficient Generation of High-Quality Atomic Charges. AM1-BCC Model: I: Method**, *Journal of Computational Chemistry*, Vol. 21, pp. 132-146, **2000**.
- [Jakalian-2002] Araz Jakalian, David B. Jack and Christopher I. Bayly, **Fast, Efficient Generation of High-Quality Atomic Charges. AM1-BCC Model: II: Parameterization and Validation**, *Journal of Computational Chemistry*, Vol. 23, pp. 1623-1641, **2002**.
- [Katritzky-2000] Alan R. Katritzky and A.F. Pozharskii, **Handbook of Heterocyclic Chemistry**, Academic Press, 2nd Edition, **2000**.
- [Sayle-1999] Roger Sayle and Jack Delany, **Canonicalization and Enumeration of Tautomers**, in *Innovative Computational Applications*, Institute for International Research, Sir Francis Drake Hotel, San Francisco, 25-27 October **1999**.
- [McGuire-2000] A. Ting, R. McGuire, A.P. Johnson and S. Green, **Expert System Assisted Pharmacophore Identification**, *Journal of Chemical Information and Computer Science (JCICS)*, Vol. 40, No. 2, pp. 347-353, **2000**.
- [Trepalin-2003] Sergey V. Trepalin, Andrey V. Skorenko, Konstantin V. Balakin, Anatoly F. Nasonov, Stanley A. Lang, Andrey A. Ivashchenko and Nikolay P. Savchuk, **Advanced Exact Structure Searching in Large**

Databases of Chemical Compounds, *Journal of Chemical Information and Computer Science (JCICS)*, Vol. 43, No. 3, pp. 852–860, **2003**.

[Yang-1993] Yang, A.-S., M. R. Gunner, R. Sampogna, K. Sharp and B. Honig, **On the Calculation of pKa's in Proteins**, *Proteins*, Vol. 15, pp. 252-265, **1993**.

INDEX

Symbols

-all
command line option, 8

-altlocs
command line option, 14

-am1
command line option, 14

-am1bcc
command line option, 15

-amberff94
command line option, 15

-can
command line option, 8

-ch3
command line option, 8

-clear,-none
command line option, 15

-count
command line option, 8, 10

-formal
command line option, 15

-gasteiger
command line option, 15

-in
command line option, 7, 10, 12, 14

-initial
command line option, 15

-kekule
command line option, 8

-level
command line option, 8

-max
command line option, 8, 10

-mmff
command line option, 15

-noh
command line option, 15

-opls
command line option, 15

-out
command line option, 8, 11, 12, 14

-param
command line option, 8, 11, 15

-paramfile
command line option, 8, 11, 15

-prefix
command line option, 8, 11, 15

-reasonable
command line option, 8

-savestereo
command line option, 9

-singlePoint
command line option, 15

-uniq
command line option, 9

A

APPNAME_OE_ARCH, 4

C

command line option

- all, 8
- altlocs, 14
- am1, 14
- am1bcc, 15
- amberff94, 15
- can, 8
- ch3, 8
- clear,-none, 15
- count, 8, 10
- formal, 15
- gasteiger, 15
- in, 7, 10, 12, 14
- initial, 15
- kekule, 8
- level, 8
- max, 8, 10
- mmff, 15
- noh, 15
- opls, 15
- out, 8, 11, 12, 14
- param, 8, 11, 15

- paramfile, 8, 11, 15
- prefix, 8, 11, 15
- reasonable, 8
- savestereo, 9
- singlePoint, 15
- uniq, 9

E

environment variable

- APPNAME_OE_ARCH, 4
- OE_ARCH, 4
- OE_LICENSE, 3
- PATH, 4, 5

O

- OE_ARCH, 4
- OE_LICENSE, 3

P

- PATH, 4, 5