
QuacPac
Quality Atomic Charges,
Proton Assignment and
Canonicalization

version 1.3.1

OpenEye Scientific Software, Inc.

July 3, 2008

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

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

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

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

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

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

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

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

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

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

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

CONTENTS

1	Introduction to QuacPac	1
2	Tautomers - Enumeration and Canonicalization	2
2.1	Introduction	2
2.2	Usage	2
3	Pkatyper - Ligand pKa	6
3.1	Introduction	6
3.2	Theory	6
3.3	Usage	7
4	Molcharge - Partial Charges	9
4.1	Introduction	9
4.2	Theory	10
4.3	Usage	11
5	Application Installation	13
6	QuacPac Library	14
6.1	Function API	14
6.2	Functor API	15
6.3	OECharges Namespace	16
6.4	Example Program	17
7	Molecular File Formats	21
8	Release Notes	22
8.1	QuacPac 1.3.1	22
8.2	QuacPac 1.3.0	22
8.3	QuacPac 1.1.0	23
A	Bibliography	24

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.

Tautomers - Enumeration and Canonicalization

2.1 Introduction

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 coordinateless. 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 from the output from tautomers. We recommend that in preparation of small-molecules for study, charge-state and tautomer enumeration be performed before conformer generation.

2.2 Usage

2.2.1 Command Line Interface

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

```
prompt> tautomers --help all
```

will generate the following output:

```
Complete parameter list
```

```
tautomer
  -all : Enumerate all (up to level 5) tautomers
  -can : Write tautomers in canonical SMILES
  -ch3 : Tautomerize alpha methyl/methylene groups
  -count : Only count rather than generate the states
  -in : Input filename
  -kekule : Write Kekule structures
  -level : Acceptable pseudo-energy level of tautomers
  -max : Maximum number of tautomers per molecule
  -out : Output filename
  -param : parameter file of tautomers settings
  -paramfile : Parameter filename for output
  -prefix : Prefix to use to name output files
  -reasonable : Returns a reasonable looking unique tautomer
  -uniq : Return a single unique tautomer
```

Command line options are distinguished from real filenames by having a '-' prefix. Options can appear anywhere on the command line, *i.e.* before, after or in between filenames. When incompatible options are specified the last one given on the command line takes effect.

The first filename given on the command line is taken to be the input file, and the optional second filename is treated as the output file. A minus character may be used in place of the input filename to specify that the input is to be read from standard input, stdin, and in place of the output filename to specify that the output is to be written to standard output, stdout. If only one filename is specified on the command line, the output is written to stdout by default.

2.2.2 Command Line Options

- 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 5` and `-ch3`.
- a** Synonym for `-all`.
- 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.
- c** Synonym for `-can`.
- ch3** Include tautomerisation across methyl and methylene groups that are alpha to a conjugated system. 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.
- count** Output just the number of tautomers per compound rather than listing the tautomers.
- C** Synonym for `-count`.
- in** The input filename for reading in molecules.
- kekule** Output is in Kekulé format for `.smi`, `.ism` and `.can`. Often, disabling aromaticity makes it easier to understand the differences between tautomers.
- level *n*** Manually set the acceptable approximate tautomer energy level to use in enumeration. This takes an integer value between zero and six inclusive, where zero corresponds to lowest energy states and six corresponds

to the highest energy state. The `-all` command line option increases this value to at least five. 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.

-l Synonym for `-level`.

-max *n* 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.

-out Output filename, where the file extension indicates the output format.

-param This can be used to name an input parameter file of tautomers settings, especially useful for running the program with similar flags as a previous run. Flags coming after `-param` override parameters set by the parameter file.

-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’`.

-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.

-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.

-u Synonym for `-uniq`.

2.2.3 Example Executions

Consider the following input file, `guanine.smi`, that contains just the following line.

```
c1[nH]c2c(=O)[nH]c(nc2n1)N
```

The following command can be used to enumerate all of the reasonably low energy tautomers of this structure.

```
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 -a guanine.smi output.smi
```

which will write a total of 300 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 -count -all guanine.smi  
300
```

Finally, the canonicalization abilities of the `tautomers` program can be assessed by canonicalizing all 300 of the guanine tautomers generated above. The example below uses the UNIX command line utility “`uniq`” which counts the number of repeated lines in a file. By piping the output of `tautomers` with the `-u` flag to UNIX command `uniq`, the output below confirms we generate 300 identical copies of the SMILES string “`c1[nH]c2c(=O)nc([nH]c2n1)N`”.

```
prompt> tautomers -u output.smi | /usr/bin/uniq -c  
300 c1[nH]c2c(=O)nc([nH]c2n1)N
```

Pkatyper - Ligand pKa

3.1 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 above the aqueous ground-state can be important to biological processes. It is appropriate to also 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 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 ΔG . This value was chosen to correspond to the similar range that is often used for generating conformational ensembles of small molecules.

3.2 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 or 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 coordinateless. When used on molecules with three-dimensional coordinates, pkatyper attempts to place new hy-

drogens 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 from 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.

3.3 Usage

3.3.1 Command Line Interface

Executing pkatyper with no arguments will result in:

```
prompt> pkatyper

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

Parameters

For a complete listing of parameters use the “--help all” flag as follows.

```
prompt> pkatyper --help all
```

- 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.
- 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. A description of supported file formats can be found in the final chapter of this document.
- 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** This can be used to name an input parameter file of pkatyper settings, especially useful for running the program with similar flags as a previous run. Flags coming after -param override parameters set by the parameter file.

-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’`.

3.3.2 Molecular formats

pkatypeper can read and write a variety of molecular formats. For details, please see the chapter on molecular formats below.

3.3.3 Example executions

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

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

Reads the file `drugs.sdf` in `sd` format and 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> pkatypeper -in drugs.sdf -out enumerated_drugs.smi
```

This command generates the exact same behavior as described above.

```
prompt> pkatypeper drugs.sdf enumerated_drugs.sdf
```

Reads the file `drugs.sdf` in `sd` format and enumerates the pKa states of each molecule and writes them into the file `enumerated_drugs.sdf` in `sd` format. Molecules with only one identified pKa state are passed through to the output file. Explicit protons will be added where appropriate, and a rule-based method is used to generate coordinates for the proton. However, the *heavy-atom coordinates of the molecule are not perturbed*, so in cases where the change in protonation-state would lead to a modified heavy-atom geometry, the user must follow-up with a re-generation of the conformation in order to get completely reasonable proton and heavy-atom positions.

```
prompt> pkatypeper -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.

Molcharge - Partial Charges

4.1 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 chemoinformatics, 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 $\text{N}[\text{C}^+](\text{N})\text{N}$ with the formal charge assigned to the carbon, or as $[\text{NH}_2^+]=\text{C}(\text{N})\text{N}$ with the formal charge assigned to an arbitrarily to one of the otherwise equivalent nitrogens. A similar example is a thiocarboxylate group, where either $\text{C}(=\text{O})[\text{S}^-]$ or $\text{C}(=\text{S})[\text{O}^-]$ 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.

4.2 Theory

4.2.1 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.

4.2.2 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.

4.2.3 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.*

4.2.4 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 MMFF94 absolute values 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 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.

4.3 Usage

4.3.1 Command Line Interface

Executing molcharge with no arguments will result in:

```
prompt> molcharge

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

For a complete listing of parameters, use the argument “-help all”

Command-Line Options

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.

- am1** Assign AM1 partial charges to each atom.
- am1bcc** Assign AM1-BCC partial charges to each atom.
- clear,-none** Set the partial charges to zero.
- formal** Set the partial charges of each atom to its formal charge. Unlike the `-none` command line option, this at least preserves the net charge on the molecule.
- gasteiger** Assign Marsili-Gasteiger partial charges to each atom.
- in** This is the input file. This file can contain molecules in a wide-variety of molecular formats. For further details please see the chapter discussing them below. 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.

- initial** Set the charges to the Gasteiger seed charges.
- mmff** Assign MMFF94 partial charges to each atom.
- 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.
- 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.
- out** This is the output file and it is a required parameter. Since this 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`. For further information on molecular formats please see the chapter discussing them below. 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.
- param** This can be used to name an input parameter file of molcharge settings, especially useful for running the program with similar flags as a previous run. Flags coming after `-param` override parameters set by the parameter file.
- 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'`.
- singlePoint** Do not optimize the coordinates under the AM1 potential before calculating the charges. [default = optimize]

4.3.2 Example executions

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.

Application Installation

5.0.3 Linux/Unix

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

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

5.0.4 Windows

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

5.0.5 OS X

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

QuacPac Library

The QuacPac library has a very small API with two functors and three free functions. This API gives access to tautomer and pKa-state enumeration and partial charging. There is one tautomer functor used as a call-back function in association with tautomer enumeration and there is one pKa functor used analogously as a call-back function in association with pKa enumeration. For basic information on functors, please refer to the OEChem Theory Manual. For both of these enumeration functions, each time a new molecule is generated, the functor's operator() is called with the new molecule. The enumeration process continues until the functor's operator() returns false or the enumeration process completes. This API allows complete user specification of the termination criteria for both tautomer and pKa enumeration. Finally, a single free function, `OEAssignPartialCharges` provides access to all of OpenEye's partial charging models. The namespace `OECharges` provides a series of unsigned integer tags (e.g. `OECharges::MMFF`) are passed to the free function to determine which charge model is used.

Any executable built with the QuacPac library will need to link the three OEChem libraries (oechem, oesystem, and oeplatform). Thus, if you have OEChem and Omega, you will be able to use the Omega library.

The API of the library in this release is subject to change.

6.1 Function API

```
unsigned int OEEnumerateFormalCharges(OEChem::OEMolBase &mol,
                                     OEMolFunctionBase &mf,
                                     bool verbose = false);

unsigned int OEEnumerateTautomers(OEChem::OEMolBase &mol,
                                 OEMolFunctionBase &mf,
                                 unsigned int allflag = 0,
                                 bool ch3flag = false);

bool OEAssignPartialCharges(OEChem::OEMolBase &mol,
                            unsigned int method = OECharges::Default,
                            bool noHydrogen = false,
                            bool debug = false);

bool OEAssignPartialCharges(OEChem::OEMCMolBase &mol,
                            unsigned int method = OECharges::Default,
                            bool noHydrogen = false,
                            bool debug = false);
```

6.1.1 OEEEnumerateFormalCharges

This function has three arguments. The first is the non-const molecule whose formal charges are to be enumerated. For details of the enumeration process, please see the section concerning `pkatyper` above. The second argument is a functor call-back (see functor API below). The functor's `operator()` is called with each new protonation state enumerated. The enumeration will continue until the functor returns false or until the enumeration is complete. The third argument determines if verbose atom-type information should be written to standard out. The function will return the total number of states which were enumerated.

6.1.2 OEEEnumerateTautomers

This function has four arguments. The first argument is a non-const molecule that will be the basis for the tautomer enumeration. The second argument is a functor call-back (see functor API below). The functor's `operator()` is called with each new tautomer state enumerated. The enumeration will continue until the functor returns false or until the enumeration is complete. The third argument is an unsigned int which indicates the maximum acceptable energetic category of enumerated tautomers. The function will return all categories of tautomers from the lowest available up to this cutoff. If the only available level is higher than the cutoff, that single level of tautomers will be enumerated. This control is in addition to the call-back control. Finally, the Boolean `ch3flag` argument controls whether tautomer enumeration should include tautomerization of methyl and methylene groups adjacent to a conjugated system. This allows cyclohexa-2,4-dien-1-one, O=C1CC=CC=C1, to be canonicalized as a tautomer of phenol, Oc1ccccc1. Unfortunately, the combinatorial explosion from tautomerizing across the alpha carbon of amino acids means that this option should not be used when enumerating the tautomers of proteins and large peptides. This function returns the total number of tautomers enumerated.

6.1.3 OEAssignPartialCharges

This function sets the partial charge value of each atom on a molecule using the `OEAtomBase::SetPartialCharge` method. The molecule to be charged is passed as the first argument. The second argument is the charge model to be used. These values should be selected from the `OECharge` namespace (vida infra). The third argument is a boolean for whether the final molecule should have any explicit hydrogens. If `noHydrogen` is set to true, all hydrogens will be made implicit, and the charge on each heavy atom with attached hydrogens will be adjusted to be the sum of its original partial charge and all of the attached hydrogen partial charges. The final flag is the debug flag which controls the volume of debug information written to standard error. The function returns a boolean value evaluating the success of the calculation.

When calling this function with an `OEAtomBase` and `AM1BCC` charges, the multiconformer algorithm described in the theory section is performed.

6.2 Functor API

```
typedef OESystem::OEUnaryFunction<OEChem::OEMolBase,bool,true,false>
                                         OEMolFunctionBase;

class OETyperMolFunction : public OEMolFunctionBase
{
public:
```

```

OETyperMolFunction(OEChem::oemolostream &ofp, bool arom,
                  bool ct=false, unsigned int max = 0);
void Reset();
bool operator () (const OEChem::OEMolBase &inmol);
base_type *CreateCopy() const;
};

class OETautomerMolFunction : public OEMolFunctionBase
{
public:
    OETautomerMolFunction(OEChem::oemolostream &ofp, bool arom,
                          bool ct=false, unsigned int max = 0, bool mostAro = false);
    unsigned int GetCount() const;
    const OEChem::OEMolBase& GetMolecule() const;
    void Reset();
    bool operator () (const OEChem::OEMolBase &inmol);
    base_type *CreateCopy() const;
};

```

6.2.1 OEMolFunctionBase

This is a simple typedef of a specific type of OEUnaryFunction. The first two template arguments indicate that an OEMolFunctionBase will have an operator() that takes an OEChem::OEMolBase as an argument and will return boolean value. The final two template arguments indicate that the OEChem::OEMolBase argument will be const but that the operator() function will not be const respectively. For more information on OEUnaryFunctions, please see the OESystem documentation.

6.2.2 OETyperMolFunction and OETautomerMolFunction

Currently these two example OEMolFunctionBases are similar, however it is expected that they will diverge more over time. They share the same first four arguments. The first argument is the oemolstream where the enumerated molecules will be placed. The second argument is a boolean indicating whether aromaticity should be calculated for the enumerated structure. The third argument is a boolean indicating whether the enumerated states should only be counted (rather than actually listed). The fourth argument is an unsigned int indicating the maximum number of states that should be enumerated for any single input molecule. The final argument for OETautomerMolFunction states whether a '-reasonable' type calculation is being setup and only the most aromatic tautomer should be stored. After the calculation, the most aromatic tautomer can be retrieved using GetMolecule. When using the same OETautomerMolFunction for multiple tautomer runs, remember to call Reset in between each one.

Each class has a Reset () function that can be called to reset the output counter.

6.3 OECharges Namespace

```

namespace OECharges {
static const unsigned int Default      = 0;
static const unsigned int None        = 1;
static const unsigned int Formal      = 2;
static const unsigned int Initial     = 3;
static const unsigned int Gasteiger   = 4;
}

```

```

static const unsigned int MMFF94      = 5;
static const unsigned int AM1BCC      = 6;
static const unsigned int AM1BCCSPt  = 7;
static const unsigned int OPLS       = 8;
static const unsigned int AM1        = 9;
static const unsigned int AM1SPt     = 10;
}

```

Each of these partial charge methods is dependent on the formal charges and protonation states of the ligand already being determined.

OECharges::Default The current default charges or MMFF.

OECharges::None Removed all partial charges.

OECharges::Formal Copies the formal charge field of atoms into the partial charge field.

OECharges::Initial Smears unit charges in the partial charge field onto resonance shared atoms.

OECharges::Gasteiger Assigns Gasteiger partial charges.

OECharges::MMFF94 Assigns MMFF94 partial charges based.

OECharges::AM1BCC AM1 charges with bond-charge correction.

OECharges::AM1BCCSPt AM1BCC single-point calculation.

OECharges::OPLS OPLS protein dictionary charges.

OECharges::AM1 AM1 Mulliken charges.

OECharges::AM1SPt AM1 single-point calculation.

6.4 Example Program

The following code example is a simple example of how to use the oe proton library provided in QuacPac to assign partial charges to an OEGraphMol. The program opens the file foo.sdf and reads all of the molecules in the file. Each molecule has its partial charge calculated with the AM1BCC method (including AM1 optimization). Each hydrogen is then made to be implicit, and it's partial charge is added to the partial charge on it's parent heavy-atom. Finally, each molecule is written to the file foo.mol2.

```

1  /*****
2  Copyright (C) 2004,2007, 2008 by OpenEye Scientific Software, Inc.
3  *****/
4  #include "oechem.h"
5  #include "oesystem.h"
6  #include "oequacpac.h"
7
8  using namespace OESystem;
9  using namespace OEChem;
10 using namespace OEProton;
11
12 int main(int argc, char *argv[])
13 {
14     if (argc!=3)
15         OEThrow.Usage("%s <mol-infile> <mol-outfile>", argv[0]);
16

```

```

17 oemolistream ifs(argv[1]);
18 if(!ifs)
19     OThrow.Error("Unable to open %s for reading", argv[1]);
20
21 oemolostream ofs(argv[2]);
22 if(!ofs)
23     OThrow.Error("Unable to open %s for writing", argv[2]);
24
25 OEGraphMol mol;
26 const bool noHydrogen = true;
27 const bool debug = false;
28 while(OEReadMolecule(ifs,mol))
29 {
30     OEAssignPartialCharges(mol,OECharges::MMFF94,noHydrogen,debug);
31     OEWwriteMolecule(ofs,mol);
32 }
33
34 return 0;
35 }

```

Listing 6.1: Calculating Partial Charges

This second example shows how to enumerate pKa states of a molecule. Molecules are read from the input file `foo.smi`. The `OETyperMolFunction` is created so that a default output stream (`std::out`, SMILES format) is used, aromaticity will be called, compounds are enumerated rather than only being counted, and a maximum of 200 states per molecule will be generated. The state counter is reset for each new molecule with the `OETyperMolFunction::Reset` function.

```

1  /*****
2  Copyright (C) 2004,2007, 2008 by OpenEye Scientific Software, Inc.
3  *****/
4  #include <iostream>
5  #include "oechem.h"
6  #include "oesystem.h"
7  #include "oequacpac.h"
8
9  using namespace OESystem;
10 using namespace OEChem;
11 using namespace OEProton;
12
13 int main(int argc, char *argv[])
14 {
15     if (argc!=3)
16         OThrow.Usage("%s <mol-infile> <mol-outfile>", argv[0]);
17
18     oemolistream ifs(argv[1]);
19     if(!ifs)
20         OThrow.Error("Unable to open %s for reading", argv[1]);
21
22     oemolostream ofs(argv[2]);
23     if(!ofs)
24         OThrow.Error("Unable to open %s for writing", argv[2]);
25
26     OEGraphMol mol;
27     const bool aromatic = true;
28     const bool countOnly = false;
29     const unsigned int maxCount = 100;
30     OETyperMolFunction tmf(ofs,aromatic,countOnly,maxCount);
31     const bool verbose = false;
32     while(OEReadMolecule(ifs,mol))
33     {
34         OEEnumerateFormalCharges(mol,tmf,verbose);
35         tmf.Reset();

```

```
36 }
37
38 return 0;
39 }
```

Listing 6.2: Enumerating Ionization States

This third example listing expands on the previous example. Here, rather than enumerate the pKa states to `std::out`, they are enumerated into a stringstream. Each enumerated pKa state is then passed into tautomer enumeration. In the tautomer enumeration phase, the number of states are only counted rather than being enumerated. In both phases, the enumeration is capped at 200 states per molecule. The states of both the ionization state counter and the tautomer counter are reinitialized with their `Reset` functions.

```
1  /*****
2  Copyright (C) 2004,2007, 2008 by OpenEye Scientific Software, Inc.
3  *****/
4  #include <iostream>
5  #include "oeplatform.h"
6  #include "oesystem.h"
7  #include "oechem.h"
8  #include "oequacpac.h"
9
10 using namespace OEPlatform;
11 using namespace OESystem;
12 using namespace OEChem;
13 using namespace OEProton;
14 using namespace std;
15
16 int main(int argc, char *argv[])
17 {
18     if (argc!=2)
19         OEThrow.Usage("%s <mol-infile>", argv[0]);
20
21     oemolistream ifs(argv[1]);
22     if(!ifs)
23         OEThrow.Error("Unable to open %s for reading", argv[1]);
24
25     oemolostream nulfs(&oenu, false);
26     if(!nulfs)
27         OEThrow.Error("Unable to declare oenu molstream.");
28
29     OEGraphMol mol,pkaState;
30     oemolostream ofs;
31     const bool aromatic = true;
32     const bool tyCountOnly = false;
33     const unsigned int maxCount = 200;
34     OETyperMolFunction tymf(ofs,aromatic,tyCountOnly,maxCount);
35     const bool verbose = false;
36
37     const bool taCountOnly = true;
38     OETautomerMolFunction tamf(nulfs,aromatic,taCountOnly,maxCount);
39     oemolistream iss;
40     unsigned int count;
41     while(OEReadMolecule(ifs,mol))
42     {
43         //enumerate pka states
44         ofs.openstring();
45         OEEnumerateFormalCharges(mol,tymf,verbose);
46         tymf.Reset();
47         iss.openstring(ofs.GetString());
48
49         //count tautomers of pka states
50         count = 0;
```

```
51 while(OEReadMolecule(iss,pkaState))
52 {
53     count += OEEnumerateTautomers(pkaState,tamf);
54     tamf.Reset();
55 }
56 cerr << count << " tautomer/pka states for " << mol.GetTitle() << endl;
57 }
58
59 return 0;
60 }
```

Listing 6.3: Counting Tautomers

Molecular File Formats

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

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

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

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

```
prompt> (quacpac application) -in - -out -
```

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

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

```
prompt> (quacpac application) -in .ism -out .oeb.gz
```

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

Release Notes

8.1 QuacPac 1.3.1

8.1.1 New Features

1. 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.

8.1.2 Bug fixes

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

8.2 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.

8.2.1 Bug fixes

1. Library has changed names from `liboeiproton` to `liboequacpac`
2. Memory leak related to `OETyperMolFunction` and `OETautomerMolFunction` is fixed.
3. SMILES map indexes and names are no longer lost when outputting molecules.

8.2.2 New features

1. A reasonable looking tautomer may be calculated by setting the `mostAro` boolean to `true` in `OETautomerMolFunction`. After the `max` number of tautomers have been attempted, the most aromatic looking molecule may be retrieved using the `GetMolecule` method.
2. An `OEMCMolBase` version of `OEAssignPartialCharges` has been added for calculating multiconformer AM1BCC charges

8.3 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 `oeiproton` library. The `oeiproton` 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.

The two applications of the preps package allow calculation of a polynomial representation of the electrostatic potential map around a small molecule as well as conversion between these polynomial representations and the more common grid representations. While preps package is solid code, it has not been tested by nearly as many users as the packages mentioned above.

Finally, `qpd` is an application for calculating electrostatic descriptors of molecules. Like the preps package, `qpd` is believed to be solid code, but has not been tested as thoroughly as the first four applications.

Bibliography

1. E. Alexov and M.R. Gunner, "Incorporating Protein Conformational Flexibility into pH-titration Calculations: Results on T4 Lysozyme", *Biophys. J.*, Vol. 74, pp. 2075-2093, 1999.
2. John W. Baker, "Tautomerism", D. Van Nostrand Company Inc. Publishers, 1934.
3. 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 (JACS)*, Vol. 107, pp. 3902-3909, 1985.
4. José Elguero, Claude Marzin, Alan R. Katritzky and Paolo Linda, "The Tautomerism of Heterocycles", Supplement 1, *Advances in Heterocyclic Chemistry*, Academic Press, 1976.
5. 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.
6. J. Gasteiger and M. Marsili, "A New Model for Calculating Atomic Charges in Molecules", *Tetrahedron Letters*, pp. 3181-3184, 1978.
7. J. Gasteiger and M. Marsili, "Iterative Partial Equalization of Orbital Electronegativity - A Rapid Access to Atomic Charges", *Tetrahedron*, Vol. 36, pp. 3219-3228, 1980.
8. 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.
9. 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.
10. 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.
11. Alan R. Katritzky and A.F. Pozharskii, "Handbook of Heterocyclic Chemistry", Academic Press, 2nd Edition, 2000.
12. 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.
13. 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.

14. 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.
15. 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.