



OpenEye
Scientific Software

QuacPac TK – C++
Release 1.5.0

OpenEye Scientific Software, Inc.

January 11, 2012

CONTENTS

1	Front Matter	1
2	Quacpac Theory	3
2.1	Introduction to lproductl	3
2.2	<i>tautomers</i> - Enumeration and Canonicalization	3
2.3	<i>pkatyper</i> - Ligand pKa	3
2.4	<i>molcharge</i> - Partial Charges	4
3	Toolkit	7
3.1	Quacpac Library	7
4	API	13
4.1	OEProton Classes	13
4.2	OEProton Constants	15
4.3	OEProton Functions	16
5	Bibliography	19
5.1	Bibliography	19
6	Release Notes	21
6.1	QuacPac TK 1.5.1	21
6.2	QuacPac TK 1.5.0	21
6.3	QuacPac TK 1.4.1	21
6.4	QuacPac TK 1.4.0	22
6.5	QuacPac TK 1.3.1	22
6.6	QuacPac TK 1.3.0	22
6.7	QuacPac TK 1.1.0	23
6.8	Indices and tables	23
	Bibliography	25
	Index	27

FRONT MATTER

Copyright 1997-2012 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.

QUACPAC THEORY

2.1 Introduction to |product|

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. |product| 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.

2.2 *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.

2.3 *pkatyper* - Ligand pKa

2.3.1 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/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.

2.3.2 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 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.

2.4 *molcharge* - Partial Charges

2.4.1 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 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 $\text{N}[\text{C}^+](\text{N})\text{N}$ 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 cheminformatics. 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.

2.4.2 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.

TOOLKIT

3.1 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 OECheges provides a series of unsigned integer tags (e.g. OECheges::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.

3.1.1 Quacpac Examples

The following code example is a simple example of how to use the OEProton 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 its partial charge is added to the partial charge on its parent heavy-atom. Finally, each molecule is written to the file foo.mol2.

Listing 1: Calculating Partial Charges

```
/*  
  Copyright (C) 2004, 2007, 2008, 2009 by OpenEye Scientific Software, Inc.  
*/  
#include "oechem.h"  
#include "oesystem.h"  
#include "oequacpac.h"  
  
using namespace OESystem;  
using namespace OEChem;  
using namespace OEProton;
```

```

int main(int argc, char *argv[])
{
    if (argc!=3)
        OThrow.Usage("%s <mol-infile> <mol-outfile>", argv[0]);

    oemolistream ifs(argv[1]);
    if(!ifs)
        OThrow.Error("Unable to open %s for reading", argv[1]);

    oemolostream ofs(argv[2]);
    if(!ofs)
        OThrow.Error("Unable to open %s for writing", argv[2]);

    OEGraphMol mol;
    const bool noHydrogen = true;
    const bool debug = false;
    while (OEReadMolecule(ifs,mol))
    {
        OEAssignPartialCharges(mol,OECharges::MMFF94,noHydrogen,debug);
        OEWriteMolecule(ofs,mol);
    }

    return 0;
}

```

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.

Listing 2: Enumerating Ionization States

```

/*****
  Copyright (C) 2004, 2007, 2008, 2009 by OpenEye Scientific Software, Inc.
  *****/
#include <iostream>
#include "oechem.h"
#include "oesystem.h"
#include "oequacpac.h"

using namespace OESystem;
using namespace OEChem;
using namespace OEProton;

int main(int argc, char *argv[])
{
    if (argc!=3)
        OThrow.Usage("%s <mol-infile> <mol-outfile>", argv[0]);

    oemolistream ifs(argv[1]);
    if(!ifs)
        OThrow.Error("Unable to open %s for reading", argv[1]);

    oemolostream ofs(argv[2]);
    if(!ofs)

```

```

    OThrow.Error("Unable to open %s for writing", argv[2]);

    OEGraphMol mol;
    const bool aromatic = true;
    const bool countOnly = false;
    const unsigned int maxCount = 100;
    OETyperMolFunction tmf(ofs,aromatic,countOnly,maxCount);
    const bool verbose = false;
    while(OEReadMolecule(ifs,mol))
    {
        OEEnumerateFormalCharges(mol,tmf,verbose);
        tmf.Reset();
    }

    return 0;
}

```

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.

Listing 3: Counting Tautomers

```

/*****
  Copyright (C) 2004, 2007, 2008, 2009 by OpenEye Scientific Software, Inc.
  *****/
#include <iostream>
#include "oeplatform.h"
#include "oesystem.h"
#include "oechem.h"
#include "oequacpac.h"

using namespace OEPlatform;
using namespace OESystem;
using namespace OEChem;
using namespace OEProton;
using namespace std;

int main(int argc, char *argv[])
{
    if (argc!=2)
        OThrow.Usage("%s <mol-infile>", argv[0]);

    oemolistream ifs(argv[1]);
    if(!ifs)
        OThrow.Error("Unable to open %s for reading", argv[1]);

    oemolostream nulfs(&oenu, false);
    if(!nulfs)
        OThrow.Error("Unable to declare oenu molstream.");

    OEGraphMol mol,pkaState;
    oemolostream ofs;

```

```

const bool aromatic = true;
const bool tyCountOnly = false;
const unsigned int maxCount = 200;
OETyperMolFunction tymf(ofs,aromatic,tyCountOnly,maxCount);
const bool verbose = false;

const bool taCountOnly = true;
OETautomerMolFunction tamf(nulfs,aromatic,taCountOnly,maxCount);
oemolistream iss;
unsigned int count;
while(OEReadMolecule(ifs,mol))
{
    //enumerate pka states
    ofs.openstring();
    OEEnumerateFormalCharges(mol,tymf,verbose);
    tymf.Reset();
    iss.openstring(ofs.GetString());

    //count tautomers of pka states
    count = 0;
    while(OEReadMolecule(iss,pkaState))
    {
        count += OEEnumerateTautomers(pkaState,tamf);
        tamf.Reset();
    }
    cerr << count << " tautomer/pka states for " << mol.GetTitle() << endl;
}

return 0;
}

```

The forth example shows how to use `OESetNeutralpHModel` to set a favorable ionization state assuming pH=7.4. It is highly recommended that the most reasonable tautomer is first determined because an unfavorable tautomer can yield a poor ionization state.

Listing 4: Setting a Reasonable Ionization State

```

/*****
Copyright (C) 2004, 2007, 2008, 2009 by OpenEye Scientific Software, Inc.
*****/
#include "oechem.h"
#include "oesystem.h"
#include "oequacpac.h"

using namespace OESystem;
using namespace OEPlatform;
using namespace OEChem;
using namespace OEProton;

int main(int argc, char *argv[])
{
    if (argc!=3)
        OEThrow.Usage("%s <mol-infile> <mol-outfile>", argv[0]);

    oemolistream ifs(argv[1]);
    if(!ifs)

```

```
    OEThrow.Error("Unable to open %s for reading", argv[1]);

oemolostream ofs(argv[2]);
if(!ofs)
    OEThrow.Error("Unable to open %s for writing", argv[2]);

OEGraphMol mol;
OEGraphMol mostReasonableTautomer;

oemolostream nulfs(&oenu, false);
const bool aromatic = true;
const unsigned int maxCount = 200;
const bool CountOnly = false;
const bool mostReasonable = true;
OETautomerMolFunction tamf(nulfs, aromatic, CountOnly, maxCount, mostReasonable);

while(OEReadMolecule(ifs, mol))
{
    OEEnumerateTautomers(mol, tamf);
    mostReasonableTautomer = tamf.GetMolecule();
    OESetNeutralpHModel(mostReasonableTautomer);
    OEWriteMolecule(ofs, mostReasonableTautomer);
    tamf.Reset();
}

return 0;
}
```


4.1 OEProton Classes

4.1.1 OETautomerMolFunction

```
class OETautomerMolFunction : public OEMolFunctionBase
```

This class represents OETautomerMolFunction. Currently OETautomerMolFunction and OETyperMolFunction 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 OETautomerMolFunction::GetMolecule. When using the same OETautomerMolFunction instance for multiple tautomer runs, remember to call OETautomerMolFunction::Reset in between each run.

Constructors

```
OETautomerMolFunction(OEChem::oemolostream &ofp, bool arom, bool ct=false,  
                    unsigned int max=0, bool mostAro=false)
```

Default and copy constructors.

operator()

```
bool operator() (const OEChem::OEMolBase &inmol)
```

CreateCopy

```
base_type *CreateCopy() const
```

GetCount

```
unsigned int GetCount () const
```

Returns the number of tautomers generated. Calling `OETautomerMolFunction::Reset` will reset this number to zero.

GetMolecule

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

This method is used in conjunction with the `mostAro` flag to retrieve a reasonable looking tautomer.

Reset

```
void Reset ()
```

This method resets the count to zero and clears the stored molecule that `OETautomerMolFunction::GetMolecule` has access to.

4.1.2 OETyperMolFunction

```
class OETyperMolFunction : public OEMolFunctionBase
```

This class represents `OETyperMolFunction`. Currently `OETyperMolFunction` and `OETautomerMolFunction` 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.

Constructors

```
OETyperMolFunction(OEChem::oemolostream &ofp, bool arom, bool ct=false,  
                  unsigned int max=0)
```

Default and copy constructors.

operator()

```
bool operator () (const OEChem::OEMolBase &inmol)
```

CreateCopy

```
base_type *CreateCopy() const
```

Reset

```
void Reset()
```

This method will reset the internal counter and should be called before reusing an `OEProton:OETyperMolFunction` instance.

4.2 OEProton Constants

4.2.1 OECharges

This namespace contains constants.

Default

The current default charges, which are MMFF94.

None

Removed all partial charges.

Formal

Copies the formal charge field of atoms into the partial charge field.

Initial

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

Gasteiger

Assigns Gasteiger partial charges

MMFF94

Assigns MMFF94 partial charges based.

AM1BCC

AM1 charges with bond-charge correction.

AM1BCCSPt

AM1BCC single-point calculation.

OPLS

OPLS protein dictionary charges.

AM1

AM1 Mulliken charges.

AM1SPt

AM1 single-point calculation.

4.3 OEProton Functions

4.3.1 OEAssignPartialCharges

```
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)
```

This function sets the partial charge value of each atom in a molecule. 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 `OECharges` namespace. The third argument is a boolean for whether the final molecule should have 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 `debug` flag controls the volume of debug information written to standard error. This function return a boolean value evaluating the success of the calculation.

This function is overloaded to accept an `OEMolBase` or an `OEMCMolBase`. Currently, only AM1 and AM1BCC charges have specialized multiconformer implementations in the `OEMCMolBase` overload. For all other charges, the `OEMCMolBase` and `OEMolBase` implementations are identical.

4.3.2 OEEnumerateFormalCharges

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

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 *pkatyper - Ligand pKa*. The second argument is a functor call-back. 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.

4.3.3 OEEnumerateTautomers

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

This function has four arguments. The first argument is a non-const molecule that will be the basis for the tautomer enumeration. For details of the enumeration process, please see *tautomers - Enumeration and Canonicalization*. The second argument is a functor call-back. 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 `bool 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.

A setting of `saveStereo=true` will prevent associated atoms or bonds from taking part in tautomerization if they have stereochemistry set. Stereochemistry can naturally be lost with the creation and removal of double bonds.

4.3.4 OEQuacPacGetArch

```
const char *OEQuacPacGetArch()
```

Returns the architecture on which the release was built

4.3.5 OEQuacPacGetLicensee

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

Fills the parameter `string &licensee` with the licensee of the license file being used

4.3.6 OEQuacPacGetPlatform

```
const char *OEQuacPacGetPlatform()
```

Returns the platform on which the release of built

4.3.7 OEQuacPacGetRelease

```
const char *OEQuacPacGetRelease()
```

Returns the versions number as a `const char*` in major.minor.bugfix form

4.3.8 OEQuacPacGetSite

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

Fills the parameter `string &licensee` with the site on the license file being used

4.3.9 OEQuacPacGetVersion

```
unsigned int OEQuacPacGetVersion()
```

Returns the build date of the release.

4.3.10 OEQuacPacIsLicensed

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

Returns a `bool` indicating if a valid license for QuacPac TK has been set

4.3.11 OESetNeutralpHModel

```
bool OESetNeutralpHModel(OEChem::OEMolBase &mol)
bool OESetNeutralpHModel(OEChem::OEMCMolBase &mol)
```

This function will attempt to set `mol` to the most energetically favorable ionization state for `pH=7.4` using a rule-based system.

BIBLIOGRAPHY

5.1 Bibliography

RELEASE NOTES

6.1 QuacPac TK 1.5.1

6.1.1 Bug Fixes

- A bug has been fixed where tautomers were not guaranteed to be in a canonical form during output. In rare circumstances this could lead to tautomers with the same unique tautomer to print it in different non-canonical forms.

6.2 QuacPac TK 1.5.0

6.2.1 New Features

- 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 *ch3flag* option will only generate tautomers appropriate for the given level.
- Improvement of the *mostAro* option, particularly involving exocyclic heteroatoms.
- Added stereo preservation flag to `OEEnumerateTautomers` with a default 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.
- Improvements to pka states.
- AmberFF94 charges can now be applied to standard protein residues.

6.3 QuacPac TK 1.4.1

6.3.1 Bug Fixes

- `OESetNeutralpHModel` has been added to the Java and Python wrappers.
- Bug has been fixed in `OEAssignPartialCharges` where it sometimes did not return false after a failure to assign charges. Now this function should always return false if charges are not set.
- Fixed tetrionic acid rule in `OESetNeutralpHModel` to allow matching of aromatic bonds.

6.4 QuacPac TK 1.4.0

6.4.1 New features

- `OESetNeutralpHModel` has been added. This function 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.
- `OEEnumerateFormalCharges` now implicitly uses the `mostAro=true` parameter on the `OETautomerMolFunction` it uses when choosing a tautomer. This provides a better reference tautomer when enumerating pKa states.
- Aromaticity settings on molecules are now unchanged when `OEAssignPartialCharges` is called. 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.

6.5 QuacPac TK 1.3.1

6.5.1 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.

6.5.2 Bug fixes

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

6.6 QuacPac TK 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.

6.6.1 New features

- 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 `OETautomerMolFunction::GetMolecule` method.
- An `OEMCMolBase` version of `OEAssignPartialCharges` has been added for calculating multiconformer AM1BCC charges

6.6.2 Bug fixes

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

6.7 QuacPac TK 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 oepron library. The oepron 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.

6.8 Indices and tables

- *Index*
- *Search Page*

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

C

Constructors

OEProton::OETautomerMolFunction, 13

OEProton::OETyperMolFunction, 14

CreateCopy

OEProton::OETautomerMolFunction, 13

OEProton::OETyperMolFunction, 15

E

Example Code

quacpac_example1.cpp, 7

quacpac_example2.cpp, 8

quacpac_example3.cpp, 9

quacpac_example4.cpp, 10

G

GetCount

OEProton::OETautomerMolFunction, 14

GetMolecule

OEProton::OETautomerMolFunction, 14

O

OEProton::OEAssignPartialCharges, 16

OEProton::OECharges, 15

OEProton::OECharges::AM1, 16

OEProton::OECharges::AM1BCC, 16

OEProton::OECharges::AM1BCCSPt, 16

OEProton::OECharges::AM1SPt, 16

OEProton::OECharges::Default, 15

OEProton::OECharges::Formal, 15

OEProton::OECharges::Gasteiger, 15

OEProton::OECharges::Initial, 15

OEProton::OECharges::MMFF94, 15

OEProton::OECharges::None, 15

OEProton::OECharges::OPLS, 16

OEProton::OEEnumerateFormalCharges, 17

OEProton::OEEnumerateTautomers, 17

OEProton::OEQuacPacGetArch, 17

OEProton::OEQuacPacGetLicensee, 17

OEProton::OEQuacPacGetPlatform, 18

OEProton::OEQuacPacGetRelease, 18

OEProton::OEQuacPacGetSite, 18

OEProton::OEQuacPacGetVersion, 18

OEProton::OEQuacPacIsLicensed, 18

OEProton::OESetNeutralpHModel, 18

OEProton::OETautomerMolFunction, 13

Constructors, 13

CreateCopy, 13

GetCount, 14

GetMolecule, 14

operator(), 13

Reset, 14

OEProton::OETyperMolFunction, 14

Constructors, 14

CreateCopy, 15

operator(), 14

Reset, 15

operator()

OEProton::OETautomerMolFunction, 13

OEProton::OETyperMolFunction, 14

Q

quacpac_example1.cpp

Example Code, 7

quacpac_example2.cpp

Example Code, 8

quacpac_example3.cpp

Example Code, 9

quacpac_example4.cpp

Example Code, 10

R

Reset

OEProton::OETautomerMolFunction, 14

OEProton::OETyperMolFunction, 15