

---

# **PypKa Documentation**

***Release 2.1.2***

**Pedro B.P.S. Reis**

**Apr 21, 2021**



# CONTENT

<b>1</b>	<b>Python Package Index</b>	<b>1</b>
<b>2</b>	<b>Required Software</b>	<b>3</b>
<b>3</b>	<b>Cross Platform</b>	<b>5</b>
<b>4</b>	<b>Basic Example</b>	<b>7</b>
4.1	API . . . . .	7
4.2	CLI . . . . .	7
4.3	Mandatory Parameters . . . . .	8
<b>5</b>	<b>Main Features</b>	<b>9</b>
5.1	Easy to Install . . . . .	9
5.2	Easy to Use . . . . .	9
5.3	API & CLI . . . . .	9
5.4	Fast and accurate . . . . .	9
5.5	Flexibility of Input Structures . . . . .	9
5.6	Lipidic Systems Support . . . . .	9
<b>6</b>	<b>Main Methods</b>	<b>11</b>
<b>7</b>	<b>Input Parameters</b>	<b>13</b>
7.1	General Parameters . . . . .	13
7.2	Poisson-Boltzmann Parameters . . . . .	13
7.3	Monte Carlo Parameters . . . . .	15
<b>8</b>	<b>FF Extension</b>	<b>17</b>
8.1	Adding extra residues . . . . .	17
8.2	Add residue block to aa.rtp file . . . . .	17
8.3	Add atom types to ffnonbonded.itp . . . . .	17
8.4	Add placeholder info on DataBaseT_old.crg and DataBaseT_old.siz . . . . .	18
8.5	Run extend_db . . . . .	18
8.6	Add residue to pdb2pqr . . . . .	18
<b>9</b>	<b>Contributing</b>	<b>19</b>
9.1	Types of Contributions . . . . .	19
9.2	Get Started! . . . . .	20
9.3	Pull Request Guidelines . . . . .	20
9.4	Deploying . . . . .	20
<b>10</b>	<b>Future Work</b>	<b>23</b>

10.1	Greater Usability . . . . .	23
10.2	Faster Calculations . . . . .	23
10.3	Accuracy Improvement . . . . .	23
<b>11</b>	<b>API Reference</b>	<b>25</b>
11.1	Titration . . . . .	25
11.2	Molecule . . . . .	25
11.3	Site . . . . .	26
11.4	Tautomer . . . . .	27
<b>12</b>	<b>Pypka</b>	<b>29</b>
12.1	Availability . . . . .	29
12.2	Contacts . . . . .	29
	<b>Index</b>	<b>31</b>

## PYTHON PACKAGE INDEX

To install the latest stable release with pip do:

```
pip3 install pypka
```



## REQUIRED SOFTWARE

Pypka depends on the following software:

- Python3.6
- Python2
- gawk

For the moment DelPhi depends on the following software:

- gcc
- gfortran
- libgfortran4

These can all be installed via apt in Debian based systems:

```
apt install gawk gcc gfortran libgfortran4 python2
```



## CROSS PLATFORM

PypKa has been developed for Linux users and it does not support Windows or MacOS. However, we do provide a docker image for these users.

```
docker container run -it -v ${PWD}:/pypka pedrishi/pypka:latest
```



## BASIC EXAMPLE

### 4.1 API

A simple example of how to use Pypka as an API is provided below. In this snippet we are estimating the pKa values for all titratable sites in the `4lzt.pdb` structure downloaded from the Protein Data Bank.

```
from pypka import Titration

params = {
    'structure'      : '4lzt.pdb',
    'ncpus'         : -1,
    'epsin'         : 15,
    'ionicstr'      : 0.1,
    'pbc_dimensions': 0
    #Set the ffinput when using PDB files from simulations
    #'ffinput': 'GROMOS' # options: GROMOS, AMBER, CHARMM
}

tit = Titration(params)

pH = 7.0
for site in tit:
    state = site.getProtState(pH)[0]
    print(site.res_name, site.res_number, site.pK, state)
```

You may also try it out on a [online notebook](#).

### 4.2 CLI

The same calculation can be done via CLI by resorting to a input parameter file.

Listing 1: parameter.dat

```
structure      = 4lzt.pdb
epsin          = 15
ionicstr       = 0.1
pbc_dimensions = 0
ncpus          = -1
sites          = all
#Set the ffinput when using PDB files from simulations
#Options: GROMOS, AMBER, CHARMM
```

To execute pypka simply type one of the two:

```
pypka parameters.dat  
python3 -m pypka parameters.dat
```

## 4.3 Mandatory Parameters

### **structure**

**Type** str

the PDB filename

### **epsin**

**Type** float

internal dielectric to be used in the PB calculations.

### **ionicstr**

**Type** float

ionic strength of the medium

### **psc\_dimensions**

**Type** int

number of dimensions with periodic boundaries. 0 for solvated proteins and 2 for lipidic systems

### **ncpus**

**Type** int

number of CPUs to use in the calculations (-1 to use all available)

## MAIN FEATURES

### 5.1 Easy to Install

As seen in the [installation page](#), PypKa is easily installed from the command line.

### 5.2 Easy to Use

The [basic example](#) provided can be effortlessly adjusted to your system. Although many [parameters](#) can be modified by an advanced user, simple users can confidently use the default values.

### 5.3 API & CLI

Every interface has its pros and cons. With PypKa you can choose to run your calculations via a python API or from the command-line. In the future we will present a webserver interface as well.

### 5.4 Fast and accurate

As a Poisson-Boltzmann-based pKa predictor, PypKa provides an interesting trade-off between speed and accuracy. You can also manually set some input parameters such as `convergence`, `gsize` or `sites`, to adjust the balance between accuracy and speed as you please. Further speed gains can be achieved by taking advantage of the highly scalable multiprocessing capabilities (`ncpus`).

### 5.5 Flexibility of Input Structures

PypKa supports both the Protein Data Bank and GROMACS input format and the most popular atomistic force-fields (AMBER, CHARMM & GROMOS) as well as experimentally determined structures. These structures are preprocessed using a modified version of PDB2PQR according to the defined `ffinput`.

### 5.6 Lipidic Systems Support

It is possible to run calculations on membrane proteins, and in the future, on lipids. Currently only some lipids are supported (DMPC, POPC, POPE and cholesterol) but more will be added. To use this feature the user is required to name the lipids in their structures according to the PypKa definition.

Example POPC file

Example POPE file

Example DMPC file

Example cholesterol file

## MAIN METHODS

**getTitrableSites** (*pdb*)

Gets the all titrable sites from the pdb

**Parameters** **pdb** (*string*) – The filename a PDB file

**Returns** All titrable sites residue numbers found in the pdb file

**Return type** list

**class Titration**

Main pypka class

**getAverageProt** (*self, site, pH*)

Calculates the average protonation of a site at a given pH value

**Parameters**

- **site** (*string*) – Residue number of the site with the suffix ‘N’ or ‘C’ to indicate a N-ter or C-ter, respectively.
- **pH** (*float*) – pH value

**Returns** Average protonation of the site at the selected pH value

**Return type** float

**getProtState** (*self, site, pH*)

Indicates the most probable protonation state of a site at a given pH value

**Parameters**

- **site** (*str*) – Residue number of the site with the suffix ‘N’ or ‘C’ to indicate a N-ter or C-ter, respectively.
- **pH** (*float*) – pH value

**Returns**

A tuple with two elements. Example: (‘protonated’, 0.9)

The first element is a string indicating the most probable state of the site. It can be either ‘protonated’, ‘deprotonated’ or ‘undefined’. The ‘undefined’ state is prompted if the average protonation state is between 0.1 and 0.9.

The second element is a float of the average protonation of the site

**Return type** tuple

**getParameters ()**

Get the parameters used in the calculations

**Returns** Current state of DelPhi parameters

**Return type** string

## INPUT PARAMETERS

### 7.1 General Parameters

**structure (str)**

**Optional** No

The input filename in Protein Data Bank or GROMACS formats

**n\_cpus (int)**

**Optional** No

Number of CPUs to use in the calculations

**ionicstr (float)**

**Optional** No

Ionic strength of the medium

**ffinput='GROMOS' (str)**

**Allowed values** ('GROMOS', 'AMBER', 'CHARMM')

Forcefield of the input `structure`. GROMOS is also valid for experimentally obtained structures

**ffID='G54a7' (str)**

**Allowed values** ('G54A7')

For the time being, only GROMOS54a7 based charged and radii are allowed. In the future more forcefields will be added.

**sites='all' (str)**

CLI Syntax: `sites_A = all sites_A = 1N, 18, 35, 48, 66, 129C`

API Syntax: `sites = 'all' sites = {'A': ('1N', '18', '35', '48', '66', '129C')}` Titration(parameters, sites=sites)

**clean\_pdb=True (boolean)**

The input `structure` files are preprocessed with PDB2PQR. To skip this step set `clean_pdb` to False.

CLI Syntax: yes or no

### 7.2 Poisson-Boltzmann Parameters

**epsin (float)**

**Optional** No

Internal dielectric to be used in the PB calculations.

**pbs\_dimensions** (int)

**Optional** No

**Allowed values** (0, 2)

Number of dimensions with periodic boundaries. Use 0 for solvated proteins and 2 for lipidic systems.

**gsize=81** (int)

DelPhi number of grid points per side of the cubic lattice.

**epssol=80.0** (float)

Solvent dielectric to be used in the PB calculations.

**relpar=0.75** (float)

DelPhi relaxation parameter in non-linear iteration convergence process.

**relfac=0.75** (float)

DelPhi spectral radius.

**nonit=0** (float)

DelPhi number of non-linear iterations.

**nlit=500** (float)

DelPhi number of linear iterations.

**convergence=0.01** (float)

DelPhi convergence threshold for maximum change of potential.

**scaleM=4** (float)

DelPhi reciprocal of one grid spacing for the finer grid.

**scaleP=1** (float)

DelPhi reciprocal of one grid spacing for the coarse grid used in the focusing step.

**slice=0.05** (float)

Only used when `pbs_dimensions` is 2. Fraction of the exterior of the lipid bilayer to be replicated with periodic boundary conditions.

**pbx=False** (boolean)

DelPhi periodic boundary conditions for the x edge of the grid

**pby=False** (boolean)

DelPhi periodic boundary conditions for the y edge of the grid

**bndcon=4** (int)

**Allowed values** (1, 2, 3, 4)

DelPhi type of boundary condition.

1. Zero
2. Dipolar
3. Focusing
4. Coulombic

**precision='single'** (str)

**Allowed values** ('single', 'double')

Precision of the compiled DelPhi version being used.

## 7.3 Monte Carlo Parameters

**seed (int)**

Seed for the MC random generator

**pH='0,14' (str)**

pH value of the calculation. It can be a single value or a range.

CLI syntax:

pH = 0, 14

API syntax:

'pH': '0, 14'

**pHstep=0.25 (float)**

In case a pH range was provided pH, the pH step of said range is pHstep



## FF EXTENSION

### 8.1 Adding extra residues

PypKa supports canonical amino acids and DNA bases, as well as some lipid molecules and ions. However, for many studies these are not sufficient and one might need to manually add these extra residues/molecules. PypKa supports out-of-the-box GROMOS54a7 and CHARMM36m FFs, but this recipe is transferable to other FFs as well.

For demonstration purposes, we will add an FMOC residue to the CHARMM36m FF for which the parameters have been kindly supplied by Alexander van Teijlingen. DataBaseT.crg and DataBaseT.siz are the main files where the charges and radii of atoms need to be added. The new DataBaseT files will be generated by extend\_db.py (tmp.crg and tmp.siz) provided one follows the steps describe below. If you believe your residue could benefit other users please create a pull request on GitHub or send the new parameters to the developers directly.

### 8.2 Add residue block to aa.rtp file

The lines should adhere to GROMACS .rtp file logic. Please use unique atom types to describe your residues in order to avoid clashes with previously declared atoms. Adding the suffix \_RESIDUE to the intended atom type should be enough.

Listing 1: aa.rtp

```
# EXTRA RESIDUES

[ FMO ] ; - Alexander van Teijlingen 07-12-2020
[ atoms ]
  C1      C1_FMO  -0.11  0
  H1      H1_FMO  0.135  1
  C2      C2_FMO  -0.11  2
  H2      H2_FMO  0.135  3
```

### 8.3 Add atom types to ffnonbonded.itp

Listing 2: ffnonbonded.itp

```
; EXTRA RESIDUES
; FMOC - Alexander van Teijlingen 07-12-2020
H1_FMO  1      1.0080  0.000  A  0.242003727796  0.12552
H2_FMO  1      1.0080  0.000  A  0.242003727796  0.12552
```

(continues on next page)

(continued from previous page)

H3_FMO	1	1.0080	0.000	A	0.242003727796	0.12552
H4_FMO	1	1.0080	0.000	A	0.242003727796	0.12552
H7_FMO	1	1.0080	0.000	A	0.242003727796	0.12552

## 8.4 Add placeholder info on DataBaseT\_old.crg and DataBaseT\_old.siz

The X.XXX placeholder marks the atom as a new addition.

Listing 3: DataBaseT\_old.crg and DataBaseT\_old.siz

```
H1 FMO X.XXX
H2 FMO X.XXX
H3 FMO X.XXX
H4 FMO X.XXX
H7 FMO X.XXX
H8 FMO X.XXX
```

## 8.5 Run extend\_db

```
python3 extend_db.py
```

New tmp.crg and tmp.siz have been generated. Please inspect them to check the new residue has been added at the bottom correctly. After the visual inspection, rename them DataBaseT.crg and DataBaseT.siz, respectively.

```
mv tmp.crg DataBaseT.crg
mv tmp.siz DataBaseT.siz
```

## 8.6 Add residue to pdb2pqr

The residue also needs to be added to PDB2PQR's database of residues, so that is not discarded on the preprocessing step.

Listing 4: /pdb2pqr/dat/CHARMM.DAT

```
# FMOC - Alexander van Teijlingen 07-12-2020
FMO C1 0.01 0.01 C1
FMO H1 0.01 0.01 H1
FMO C2 0.01 0.01 C2
FMO H2 0.01 0.01 H2
```

## CONTRIBUTING

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given. You can contribute in many ways:

### 9.1 Types of Contributions

#### 9.1.1 Report Bugs

Report bugs at <https://github.com/mms-fcul/pypka/issues>.

If you are reporting a bug, please include:

- Your operating system name and version.
- PypKa's version and parameters used for the calculation
- Origin of the input structure: Protein Data Bank or simulation (please specify the FF)

#### 9.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with “bug” and “help wanted” is open to whoever wants to implement it.

#### 9.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with “enhancement” and “help wanted” is open to whoever wants to implement it.

#### 9.1.4 Write Documentation

PypKa could always use more documentation, whether as part of the official PypKa docs, in docstrings, or even on the web in blog posts, articles, and such.

#### 9.1.5 Submit Feedback

The best way to send feedback is to file an issue at <https://github.com/mms-fcul/pypka/issues>.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.

## 9.2 Get Started!

Ready to contribute? Here's how to set up *pypka* for local development.

1. Fork the *pypka* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/pypka.git
```

3. Install your local copy into a virtualenv. Assuming you have pipenv installed, this is how you set up your fork for local development:

```
$ cd pypka/  
$ pipenv install pypka
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

Now you can make your changes locally.

5. When you're done making changes, check that your changes pass the tests:

```
$ make tests
```

6. Commit your changes and push your branch to GitHub:

```
$ git add .  
$ git commit -m "Your detailed description of your changes."  
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

## 9.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 3.5, 3.6, 3.7 and 3.8, and for PyPy.

## 9.4 Deploying

A reminder for the maintainers on how to deploy. Make sure all your changes are committed. Then run:

```
$ bump2version patch # possible: major / minor / patch
$ git push
$ git push --tags
```



## FUTURE WORK

We have built PypKa with a clear focus on three things:

- Usability
- Speed
- Accuracy

Naturally our development path also reflects these concerns.

### 10.1 Greater Usability

- Provide a Webservice Interface and platform
- More in-depth user documentation
- Improve code-level documentation
- Create a series of tutorials
- Implement a proper logging system to give the users more info
- Expand our test suite

### 10.2 Faster Calculations

- Allow the titration of a single site and all titrable residues within a cutoff radius
- Implement an asynchronous algorithm to manage the Monte Carlo runs with a dynamic step and automatic pH range
- Porting the code to python3

### 10.3 Accuracy Improvement

- Support structures with multiple chains
- Improve preprocessing guess of tautomer position guess
- Extensive benchmark and further optimization of parameters
- Support for CHARMM, AMBER and PARSE based charges and radii

- Support more lipids
- Support Nucleic Acids

## 11.1 Titration

**class** `pypka.Titration` (*parameters*, *sites='all'*, *debug=False*, *run='all'*)

Main PypKa class

**Attributes:** `params` (Config): method parameters `molecules` (dict): titrating molecules ordered by chain

**calcSiteInteractionsParallel** ()

Calculates the pairwise interaction energies

Interactions are calculated using a pool of processes and written in a formatted .dat file

**Args:** `ncpus` (int): number of cpus to be used

**calcpKint** (*unpacked\_results*)

Calculation the pKint of all tautomers.

**static getParameters** ()

Get the parameters used in the calculations.

**iterAllSitesTautomers** ()

Generator that iterates through all Tautomer instances.

The iteration is sorted by site and within each site the first to be yielded is the reference tautomer and then the rest of the tautomers by order

## 11.2 Molecule

**class** `molecule.Molecule` (*chain*, *sites*)

Molecule with more than one titrable sites

**getArrayPosition** (*atom\_id*)

Returns the index of the atom in DelPhi position array

Disclaimer: used only in testing

**getAtomsList** ()

Returns a list of atoms details

Atoms details = (id, instance, atom index in DelPhi data structures) sorted by atom id number

**getNAtoms** ()

Return number of atoms of the Site.

**getTautNAtoms** (*taut\_name*)

Return number of atoms of the tautomer named *taut\_name*

Disclaimer: it was used for testing

**getTautomerInstance** (*tautname, site\_resnum*)

Return the tautomer instance named *tautname*

Tautomer instance must exist in the site of the residue number *site\_resnum*

**iterAtoms** ()

Generator that iterates through all atoms details (name, id, position) in the Site.

**iterNonRefSitesTautomers** ()

Generator that iterates through all Tautomer instances except the reference ones.

**printAllSites** ()

Prints all Site names.

**printAllTautomers** ()

Prints all Tautomer details

## 11.3 Site

**class** `titsite.Titsite` (*res\_number, molecule*)

Titration Site with more than one Tautomer

**addAtom** (*aname, anumb*)

**Args:** *aname* (str): atom name *anumb* (int): atom id number

**addChargeSets** ()

Stores the charge set of each existing tautomer for the present Site

**addReferenceTautomer** ()

Gets last tautomer from `.sites` file adds one and saves it as the reference tautomer

**calc\_interaction\_between** (*site2*)

Calculates the pairwise interaction energies related to the two sites

**Args:** *site1* (Titsite) *site2* (Titsite)

**Ensures:** *to\_write* (str): site interaction energies formatted to be written in `.dat` file

**getAverageProt** (*pH*)

Calculates the average protonation of a site at a given pH value

**Args:** *site* (str): Residue number of the site with the suffix 'N' or 'C' to indicate a N-ter or C-ter, respectively.

*pH* (float): pH value

**Returns:** A float of the average protonation of the site at the selected pH value

**getProtState** (*pH*)

Indicates the most probable protonation state of a site at a given pH value

**Args:** *site* (str): Residue number of the site with the suffix 'N' or 'C' to indicate a N-ter or C-ter, respectively.

*pH* (float): pH value

Returns:

A tuple with two elements. Example: ('protonated', 0.9)

The first element is a string indicating the most probable state of the site. It can be either 'protonated', 'deprotonated' or 'undefined'. The 'undefined' state is prompted if the average protonation state is between 0.1 and 0.9.

The second element is a float of the average protonation of the site

**getTautomers** ()

Returns list of all tautomers instances except the tautomers of reference

**setTautomers** (*ntauts*, *resname*)

Adds Tautomers from res\_tauts to self.tautomers

**Args:** res\_tauts (list): tautomers from sites file

## 11.4 Tautomer

**class** `tautomer.Tautomer` (*name*, *site*, *molecule*)

Tautomers share the same site and atoms

Tautomers have different charge sets for the same atoms

**CalcPotentialTautomer** ()

Run DelPhi simulation of single site tautomer

**Ensures:** self.esolvation (float): tautomer solvation energy self.p\_sitpot (list): potential on site atoms

**CalcPotentialTitratingMolecule** ()

Run DelPhi simulation of the site tautomer within the whole molecule

**Ensures:** self.esolvation (float): tautomer solvation energy self.p\_sitpot (list): potential on site atoms

**calcBackEnergy** ()

Calculates background energy contribution.

**calcInteractionWith** (*tautomer2*, *site\_atom\_list*)

Calculates the interaction energy between self tautomer and tautomer2

**Args:** tautomer2 (Tautomer) site\_atom\_list (list): atom numbers belonging to the site

**calcPKint** ()

Calculates the pKint of the tautomer.

**loadChargeSet** (*res\_name*, *ref\_tautomer*)

Reads .st file related to Tautomer with residue name res\_name

Stores charge set related to both the Tautomer and the Reference Tautomer

**.st file named TYRtau1.st has charge set of TY0 and reference TY2** TYRtau2.st has charge set of TY1 and reference TY2

**setDetailsFromTautomer** ()

Set DelPhi parameters to run a calculation of a single site tautomer.

**setDetailsFromWholeMolecule** ()

Set DelPhi parameters to run a calculation of a whole molecule (all sites neutral, except one).



## PYPKA

A python module for flexible Poisson-Boltzmann based  $pK_a$  calculations.

We have implemented a flexible tool to predict Poisson-Boltzmann-based  $pK_a$  values of biomolecules. This is a free and open source project that provides a simple, reusable and extensible python API and CLI for  $pK_a$  calculations with a valuable trade-off between fast and accurate predictions. With PypKa one can enable  $pK_a$  calculations, including optional proton tautomerism, within existing protocols by adding a few extra lines of code. PypKa supports CPU parallel computing on anisotropic (membrane) and isotropic (protein) systems and allows the user to find a balance between accuracy and speed.

PypKa is written in Python and Cython and it is integrated with the Poisson-Boltzmann solver DelPhi Fortran77 via DelPhi4Py.

If you use PypKa in your research please cite the following [paper](#):

Reis, P. B. P. S., Vila-Viçosa, D., Rocchia, W., & Machuqueiro, M. (2020). PypKa: A Flexible Python Module for Poisson-Boltzmann-Based  $pK_a$  Calculations. *Journal of Chemical Information and Modeling*, 60(10), 4442–4448.

```
@article{reis2020jcim,  
author = {Reis, Pedro B. P. S. and Vila-Viçosa, Diogo and Rocchia, Walter and  
↔Machuqueiro, Miguel},  
title = {PypKa: A Flexible Python Module for Poisson-Boltzmann-Based  $pK_a$  Calculations}  
↔,  
journal = {Journal of Chemical Information and Modeling},  
volume = {60},  
number = {10},  
pages = {4442-4448},  
year = {2020},  
doi = {10.1021/acs.jcim.0c00718}  
}
```

## 12.1 Availability

PypKa can be easily installed using the pip package manager.

Source code is freely available at [GitHub](#) under the LGPL-3.0 license. The package can be installed from [PyPi](#).

## 12.2 Contacts

Please [submit a github issue](#) to report bugs and to request new features.

Alternatively you may find the developer [here](#) . Please visit our [website](#) for more information.



## A

addAtom() (*titsite.Titsite method*), 26  
 addChargeSets() (*titsite.Titsite method*), 26  
 addReferenceTautomer() (*titsite.Titsite method*),  
 26

## C

calc\_interaction\_between() (*titsite.Titsite  
 method*), 26  
 calcBackEnergy() (*tautomer.Tautomer method*), 27  
 calcInteractionWith() (*tautomer.Tautomer  
 method*), 27  
 calcpKint() (*pypka.Titration method*), 25  
 calcpKint() (*tautomer.Tautomer method*), 27  
 CalcPotentialTautomer() (*tautomer.Tautomer  
 method*), 27  
 CalcPotentialTitratingMolecule() (*tau-  
 tomer.Tautomer method*), 27  
 calcSiteInteractionsParallel() (*pypka.Titration  
 method*), 25

## G

getArrayPosition() (*molecule.Molecule method*),  
 25  
 getAtomsList() (*molecule.Molecule method*), 25  
 getAverageProt() (*Titration method*), 11  
 getAverageProt() (*titsite.Titsite method*), 26  
 getNAtoms() (*molecule.Molecule method*), 25  
 getParameters() (*pypka.Titration static method*), 25  
 getParameters() (*Titration method*), 11  
 getProtState() (*titsite.Titsite method*), 26  
 getProtStategetProtState() (*Titration  
 method*), 11  
 getTautNAtoms() (*molecule.Molecule method*), 25  
 getTautomerInstance() (*molecule.Molecule  
 method*), 26  
 getTautomers() (*titsite.Titsite method*), 27  
 getTitrableSites() (*built-in function*), 11

## I

iterAllSitesTautomers() (*pypka.Titration  
 method*), 25

iterAtoms() (*molecule.Molecule method*), 26  
 iterNonRefSitesTautomers() (*molecule.Molecule  
 method*), 26

## L

loadChargeSet() (*tautomer.Tautomer method*), 27

## M

Molecule (*class in molecule*), 25

## P

printAllSites() (*molecule.Molecule method*), 26  
 printAllTautomers() (*molecule.Molecule  
 method*), 26

## S

setDetailsFromTautomer() (*tautomer.Tautomer  
 method*), 27  
 setDetailsFromWholeMolecule() (*tau-  
 tomer.Tautomer method*), 27  
 setTautomers() (*titsite.Titsite method*), 27

## T

Tautomer (*class in tautomer*), 27  
 Titration (*built-in class*), 11  
 Titration (*class in pypka*), 25  
 Titsite (*class in titsite*), 26