# ForceBalance Developer API Guide version 1.1

Generated by Doxygen 1.7.6.1

# Contents

# 1 Project Roadmap

ForceBalance is a work in progress and is continually being improved and expanded! Here are some current and future project development ideas.

## 1.1 Short Term (summer 2013)

- Create and expand project unit testing framework to encourage a test driven development approach

- Improve/consolidate existing documentation

### 1.2 Long Term

- Development of a ForceBalance GUI interface to complement the current command line interface
- More comprehensive tutorial to walk users through the initial process of setting up targets and preparing for a successful ForceBalance run

## 2 Todo List

**Member forcebalance.abinitio.AbInitio.__init__**

Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

**Member forcebalance.abinitio.AbInitio.get_energy_force_**

Parallelization over snapshots is not implemented yet

**Member forcebalance.abinitio.AbInitio.read_reference_data**

Add an option for picking any slice out of qdata.txt, helpful for cross-validation

Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

**Member forcebalance.counterpoise.Counterpoise.loadxyz**

I should probably put this into a more general library for reading coordinates.

**Member forcebalance.forcefield.FF.mktransmat**

Only project out changes in total charge of a molecule, and perhaps generalize to fragments of molecules or other types of parameters.

The AMOEBA selection of charge depends not only on the atom type, but what that atom is bonded to.

**Member forcebalance.forcefield.FF.rsmake**

Pass in rsfactors through the input file

**Class forcebalance.gmxio.ITP_Reader**

Note that I can also create the opposite virtual site position by changing the atom labeling, woo!

**Member forcebalance.liquid.Liquid.__init__**

Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

**Member forcebalance.openmmio.OpenMM_Reader.build_pid**

Add a link here

**Member forcebalance.optimizer.Optimizer.GeneticAlgorithm**

Massive parallelization hasn't been implemented yet

**Member forcebalance.optimizer.Optimizer.Scan_Values**

Maybe a multidimensional grid can be done.

**Member forcebalance.tinkerio.Tinker_Reader.feed**

Put the rescaling factors for TINKER parameters in here. Currently we're using the initial value to determine the rescaling factor which is not very good.

**Namespace forcebalance::gmxio**

Even more stuff from forcefield.py needs to go into here.

Even more stuff from forcefield.py needs to go into here.

**Member forcebalance::gmxio.pdict**

This needs to become more flexible because the parameter isn't always in the same field. Still need to figure out how to do this.

How about making the PDIHS less ugly?

**Member forcebalance::nifty.floatornan**

I could use suggestions for making this better.

**Member forcebalance::parser.parse_inputs**

Implement internal coordinates.

Implement sampling correction.

Implement charge groups.

# 3 Namespace Index

## 3.1 Packages

Here are the packages with brief descriptions (if available):

# 4 Class Index

## 4.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

# 5 Class Index

## 5.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

# 6 File Index

## 6.1 File List

Here is a list of all files with brief descriptions:

# 7 Namespace Documentation

## 7.1 forcebalance Namespace Reference

### Packages

- namespace abinitio

  *Ab-initio fitting module (energies, forces, resp).*
- namespace abinitio_internal

  *Internal implementation of energy matching (for TIP3P water only)*
- namespace amberio

  *AMBER force field input/output.*
- namespace baseclass
- namespace basereader

  *Base class for force field line reader.*
- namespace binding

  *Binding energy fitting module.*
- namespace chemistry
- namespace contact
- namespace counterpoise

  *Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).*
- namespace custom_io

  *Custom force field parser.*
- namespace finite_difference
- namespace forcefield

  *Force field module.*
- namespace gmxio

  *GROMACS input/output.*
- namespace gmxqpio
- namespace interaction

  *Interaction energy fitting module.*
- namespace leastsq
- namespace liquid

  *Matching of liquid bulk properties.*
- namespace Mol2
- namespace mol2io

  *Mol2 I/O.*
- namespace molecule
- namespace moments

  *Multipole moment fitting module.*
- namespace nifty

  *Nifty functions, intended to be imported by any module within ForceBalance.*
- namespace objective

  *ForceBalance objective function.*
- namespace openmmio

*OpenMM input/output.*

- namespace optimizer

  *Optimization algorithms.*

- namespace parser

  *Input file parser for ForceBalance jobs.*

- namespace psi4io

  *PSI4 force field input/output.*

- namespace PT
- namespace qchemio

  *Q-Chem input file parser.*

- namespace target
- namespace tinkerio

  *TINKER input/output.*

- namespace vibration

  *Vibrational mode fitting module.*

## 7.2 forcebalance::abinitio Namespace Reference

Ab-initio fitting module (energies, forces, resp).

### Classes

- class AbInitio

  *Subclass of Target for fitting force fields to ab initio data.*

### Functions

- def weighted_variance

  *A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.*

- def weighted_variance2

  *A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.*

- def build_objective

  *This function builds an objective function (number) from the complicated polytensor and covariance matrices.*

### 7.2.1 Detailed Description

Ab-initio fitting module (energies, forces, resp).

#### Author

Lee-Ping Wang

#### Date

05/2012

### 7.2.2 Function Documentation

#### 7.2.2.1 def forcebalance.abinitio.build_objective ( *SPiXi, WCiW, Z, Q0, M0, NCP1, subtract_mean* = `True` )

This function builds an objective function (number) from the complicated polytensor and covariance matrices.

Definition at line 1151 of file abinitio.py.

#### 7.2.2.2 def forcebalance.abinitio.weighted_variance ( *SPiXi, WCiW, Z, L, R, NCP1, subtract_mean* = `True` )

A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.

Definition at line 1121 of file abinitio.py.

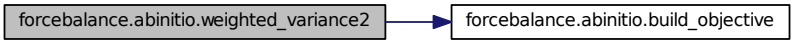
Here is the call graph for this function:


#### 7.2.2.3 def forcebalance.abinitio.weighted_variance2 ( *SPiXi, WCiW, Z, L, R, L2, R2, NCP1, subtract_mean* = `True` )

A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.

Definition at line 1135 of file abinitio.py.

Here is the call graph for this function:




## 7.3 forcebalance::abinitio_internal Namespace Reference

Internal implementation of energy matching (for TIP3P water only)

### Classes

- class AbInitio_Internal

  *Subclass of Target for force and energy matching using an internal implementation.*

### 7.3.1 Detailed Description

Internal implementation of energy matching (for TIP3P water only)

**Author**

Lee-Ping Wang

**Date**

04/2012

## 7.4 forcebalance::amberio Namespace Reference

AMBER force field input/output.

**Classes**

- class Mol2_Reader

  *Finite state machine for parsing Mol2 force field file.*
- class FrcMod_Reader

  *Finite state machine for parsing FrcMod force field file.*
- class AbInitio_AMBER

  *Subclass of Target for force and energy matching using AMBER.*

**Functions**

- def is_mol2_atom

**Variables**

- dictionary mol2_pdict = {'COUL':{'Atom':[1], 8:''}}
- dictionary frcmod_pdict

### 7.4.1 Detailed Description

AMBER force field input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

**Author**

Lee-Ping Wang

**Date**

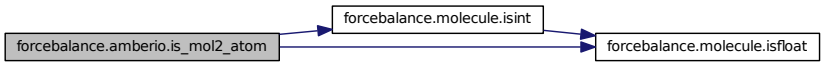
01/2012

### 7.4.2 Function Documentation

#### 7.4.2.1 def forcebalance.amberio.is_mol2_atom ( *line* )

Definition at line 32 of file amberio.py.

Here is the call graph for this function:




#### 7.4.3 Variable Documentation

##### 7.4.3.1 dictionary forcebalance::amberio::frcmod_pdict

**Initial value:**

```
1 {'BONDS': {'Atom':[0], 1:'K', 2:'B'},
2                 'ANGLES':{'Atom':[0], 1:'K', 2:'B'},
3                 'PDIHS1':{'Atom':[0], 2:'K', 3:'B'},
4                 'PDIHS2':{'Atom':[0], 2:'K', 3:'B'},
5                 'PDIHS3':{'Atom':[0], 2:'K', 3:'B'},
6                 'PDIHS4':{'Atom':[0], 2:'K', 3:'B'},
7                 'PDIHS5':{'Atom':[0], 2:'K', 3:'B'},
8                 'PDIHS6':{'Atom':[0], 2:'K', 3:'B'},
9                 'IDIHS' :{'Atom':[0], 1:'K', 3:'B'},
10                 'VDW':{'Atom':[0], 1:'S', 2:'T'}
11                 }
```

Definition at line 20 of file amberio.py.

##### 7.4.3.2 dictionary forcebalance::amberio::mol2_pdict = {'COUL':{'Atom':[1], 8:''}}

Definition at line 18 of file amberio.py.

### 7.5 forcebalance::baseclass Namespace Reference

#### Classes

- class ForceBalanceBaseClass

  *Provides some nifty functions that are common to all ForceBalance classes.*

### 7.6 forcebalance::basereader Namespace Reference

Base class for force field line reader.

#### Classes

- class BaseReader

  *The 'reader' class.*

#### 7.6.1 Detailed Description

Base class for force field line reader.

**Author**

Lee-Ping Wang

**Date**

12/2011

## 7.7 forcebalance::binding Namespace Reference

Binding energy fitting module.

**Classes**

- class BindingEnergy

  *Improved subclass of Target for fitting force fields to binding energies.*

**Functions**

- def parse_interactions

  *Parse through the interactions input file.*

### 7.7.1 Detailed Description

Binding energy fitting module.

**Author**

Lee-Ping Wang

**Date**

05/2012

### 7.7.2 Function Documentation

#### 7.7.2.1 def forcebalance.binding.parse_interactions ( *input_file* )

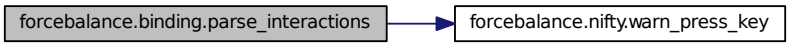
Parse through the interactions input file.

**Parameters**

| | | |
|---|---|---|
| in | *input_file* | The name of the input file. |

Definition at line 30 of file binding.py.

Here is the call graph for this function:




## 7.8 forcebalance::chemistry Namespace Reference

### Functions

- def LookupByMass
- def BondStrengthByLength

### Variables

- tuple BondEnergies = defaultdict(lambda:defaultdict(dict))
- list Radii

  *Covalent radii from Cordero et al.*
- dictionary PeriodicTable
- list Elements
- list BondChars = ['-','=','3']
- string data_from_web
- tuple line = line.expandtabs()
- tuple BE = float(line.split()[1])
- tuple L = float(line.split()[2])
- tuple atoms = re.split('[-=3]', line.split()[0])
- list A = atoms[0]
- list B = atoms[1]
- tuple bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

#### 7.8.1 Function Documentation

##### 7.8.1.1 def forcebalance.chemistry.BondStrengthByLength ( *A, B, length, artol* = 0.33, *bias* = 0.0 )

Definition at line 164 of file chemistry.py.

##### 7.8.1.2 def forcebalance.chemistry.LookupByMass ( *mass* )

Definition at line 155 of file chemistry.py.

#### 7.8.2 Variable Documentation

##### 7.8.2.1 list forcebalance::chemistry::A = atoms[0]

Definition at line 149 of file chemistry.py.

#### 7.8.2.2 tuple forcebalance::chemistry::atoms = re.split('[-=3]', line.split()[0])

Definition at line 148 of file chemistry.py.

#### 7.8.2.3 list forcebalance::chemistry::B = atoms[1]

Definition at line 150 of file chemistry.py.

#### 7.8.2.4 tuple forcebalance::chemistry::BE = float(line.split()[1])

Definition at line 146 of file chemistry.py.

#### 7.8.2.5 tuple forcebalance::chemistry::bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

Definition at line 151 of file chemistry.py.

#### 7.8.2.6 list forcebalance::chemistry::BondChars = ['-','=','3']

Definition at line 49 of file chemistry.py.

#### 7.8.2.7 tuple forcebalance::chemistry::BondEnergies = defaultdict(lambda:defaultdict(dict))

Definition at line 7 of file chemistry.py.

#### 7.8.2.8 string forcebalance::chemistry::data_from_web

Definition at line 51 of file chemistry.py.

#### 7.8.2.9 list forcebalance::chemistry::Elements

**Initial value:**

```
1 ["None",'H','He',
2             'Li','Be','B','C','N','O','F','Ne',
3             'Na','Mg','Al','Si','P','S','Cl','Ar',
4             'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge'
      ,'As','Se','Br','Kr',
5             'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn
      ','Sb','Te','I','Xe',
6             'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','
      Er','Tm','Yb',
7             'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po
      ','At','Rn',
8             'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm
      ','Md','No','Lr','Rf','Db','Sg','Bh','Hs','Mt']
```

Definition at line 40 of file chemistry.py.

#### 7.8.2.10 tuple forcebalance::chemistry::L = float(line.split()[2])

Definition at line 147 of file chemistry.py.

#### 7.8.2.11 tuple forcebalance::chemistry::line = line.expandtabs()

Definition at line 145 of file chemistry.py.

#### 7.8.2.12 dictionary forcebalance::chemistry::PeriodicTable

**Initial value:**

```
1 {'H' : 1.0079, 'He' : 4.0026,
```

```
2                 'Li' : 6.941, 'Be' : 9.0122, 'B' : 10.811, 'C' : 12.0107, 'N'
      : 14.0067, 'O' : 15.9994, 'F' : 18.9984, 'Ne' : 20.1797,
3                 'Na' : 22.9897, 'Mg' : 24.305, 'Al' : 26.9815, 'Si' : 28.0855,
       'P' : 30.9738, 'S' : 32.065, 'Cl' : 35.453, 'Ar' : 39.948,
4                 'K' : 39.0983, 'Ca' : 40.078, 'Sc' : 44.9559, 'Ti' : 47.867, '
      V' : 50.9415, 'Cr' : 51.9961, 'Mn' : 54.938, 'Fe' : 55.845, 'Co' : 58.9332,
5                 'Ni' : 58.6934, 'Cu' : 63.546, 'Zn' : 65.39, 'Ga' : 69.723, '
      Ge' : 72.64, 'As' : 74.9216, 'Se' : 78.96, 'Br' : 79.904, 'Kr' : 83.8,
6                 'Rb' : 85.4678, 'Sr' : 87.62, 'Y' : 88.9059, 'Zr' : 91.224, '
      Nb' : 92.9064, 'Mo' : 95.94, 'Tc' : 98, 'Ru' : 101.07, 'Rh' : 102.9055,
7                 'Pd' : 106.42, 'Ag' : 107.8682, 'Cd' : 112.411, 'In' : 114.818
      , 'Sn' : 118.71, 'Sb' : 121.76, 'Te' : 127.6, 'I' : 126.9045, 'Xe' : 131.293,
8                 'Cs' : 132.9055, 'Ba' : 137.327, 'La' : 138.9055, 'Ce' : 140.1
      16, 'Pr' : 140.9077, 'Nd' : 144.24, 'Pm' : 145, 'Sm' : 150.36,
9                 'Eu' : 151.964, 'Gd' : 157.25, 'Tb' : 158.9253, 'Dy' : 162.5,
      'Ho' : 164.9303, 'Er' : 167.259, 'Tm' : 168.9342, 'Yb' : 173.04,
10                 'Lu' : 174.967, 'Hf' : 178.49, 'Ta' : 180.9479, 'W' : 183.84,
      'Re' : 186.207, 'Os' : 190.23, 'Ir' : 192.217, 'Pt' : 195.078,
11                 'Au' : 196.9665, 'Hg' : 200.59, 'Tl' : 204.3833, 'Pb' : 207.2,
       'Bi' : 208.9804, 'Po' : 209, 'At' : 210, 'Rn' : 222,
12                 'Fr' : 223, 'Ra' : 226, 'Ac' : 227, 'Th' : 232.0381, 'Pa' : 23
      1.0359, 'U' : 238.0289, 'Np' : 237, 'Pu' : 244,
13                 'Am' : 243, 'Cm' : 247, 'Bk' : 247, 'Cf' : 251, 'Es' : 252, '
      Fm' : 257, 'Md' : 258, 'No' : 259,
14                 'Lr' : 262, 'Rf' : 261, 'Db' : 262, 'Sg' : 266, 'Bh' : 264, '
      Hs' : 277, 'Mt' : 268}
```

Definition at line 25 of file chemistry.py.

#### 7.8.2.13 list forcebalance::chemistry::Radii

**Initial value:**

```
1 [0.31, 0.28, # H and He
2          1.28, 0.96, 0.84, 0.76, 0.71, 0.66, 0.57, 0.58, # First row elements
3          1.66, 1.41, 1.21, 1.11, 1.07, 1.05, 1.02, 1.06, # Second row elements
4          2.03, 1.76, 1.70, 1.60, 1.53, 1.39, 1.61, 1.52, 1.50,
5          1.24, 1.32, 1.22, 1.22, 1.20, 1.19, 1.20, 1.20, 1.16, # Third row
       elements, K through Kr
6          2.20, 1.95, 1.90, 1.75, 1.64, 1.54, 1.47, 1.46, 1.42,
7          1.39, 1.45, 1.44, 1.42, 1.39, 1.39, 1.38, 1.39, 1.40, # Fourth row
       elements, Rb through Xe
8          2.44, 2.15, 2.07, 2.04, 2.03, 2.01, 1.99, 1.98,
9          1.98, 1.96, 1.94, 1.92, 1.92, 1.89, 1.90, 1.87, # Fifth row elements,
       s and f blocks
10          1.87, 1.75, 1.70, 1.62, 1.51, 1.44, 1.41, 1.36,
11          1.36, 1.32, 1.45, 1.46, 1.48, 1.40, 1.50, 1.50, # Fifth row elements,
       d and p blocks
12          2.60, 2.21, 2.15, 2.06, 2.00, 1.96, 1.90, 1.87, 1.80, 1.69]
```

Covalent radii from Cordero et al.

'Covalent radii revisited' Dalton Transactions 2008, 2832-2838.

Definition at line 10 of file chemistry.py.

## 7.9 forcebalance::contact Namespace Reference

### Functions

- def atom_distances

  *For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.*
- def residue_distances

*For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.*

#### 7.9.1 Function Documentation

##### 7.9.1.1 def forcebalance.contact.atom_distances ( *xyzlist, atom_contacts* )

For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.

xyzlist should be a traj_length x num_atoms x num_dims array of type float32

contacts should be a num_contacts x 2 array where each row gives the indices of 2 atoms whos distance you care to monitor.

Returns: traj_length x num_contacts array of euclidean distances

Note: For nice wrappers around this, see the prepare_trajectory method of various metrics in metrics.py

Definition at line 26 of file contact.py.

Here is the call graph for this function:


##### 7.9.1.2 def forcebalance.contact.residue_distances ( *xyzlist, residue_membership, residue_contacts* )

For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.

xyzlist should be a traj_length x num_atoms x num_dims array of type float32

residue_membership should be a list of lists where residue_membership[i] gives the list of atomindices that belong to residue i. residue_membership should NOT be a numpy 2D array unless you really mean that all of the residues have the same number of atoms

residue_contacts should be a 2D numpy array of shape num_contacts x 2 where each row gives the indices of the two RESIDUES who you are interested in monitoring for a contact.

Returns: a 2D array of traj_lenth x num_contacts where out[i,j] contains the distance between the pair of atoms, one from residue_membership[residue_contacts[j,0]] and one from residue_membership[residue_contacts[j,1]] that are closest.

Definition at line 85 of file contact.py.

## 7.10 forcebalance::counterpoise Namespace Reference

Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).

#### Classes

- class Counterpoise

*Target subclass for matching the counterpoise correction.*

#### 7.10.1 Detailed Description

Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE). Here we test two different functional forms: a three-parameter Gaussian repulsive potential and a four-parameter Gaussian which goes smoothly to an exponential. The latter can be written in two different ways - one which gives us control over the exponential, the switching distance and the Gaussian decay constant, and another which gives us control over the Gaussian and the switching distance. They are called 'CPGAUSS', 'CPEXPG', and 'CPGEXP'. I think the third option is the best although our early tests have indicated that none of the force fields perform particularly well for the water dimer.

This subclass of Target implements the 'get' method.

**Author**

Lee-Ping Wang

**Date**

12/2011

### 7.11 forcebalance::custom_io Namespace Reference

Custom force field parser.

**Classes**

- class Gen_Reader

  *Finite state machine for parsing custom GROMACS force field files.*

**Variables**

- list cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

  *Types of counterpoise correction.*
- list ndtypes = [None]

  *Types of NDDO correction.*
- dictionary fdict

  *Section -> Interaction type dictionary.*
- dictionary pdict

  *Interaction type -> Parameter Dictionary.*

#### 7.11.1 Detailed Description

Custom force field parser. We take advantage of the sections in GROMACS and the 'interaction type' concept, but these interactions are not supported in GROMACS; rather, they are computed within our program.

**Author**

Lee-Ping Wang

**Date**

12/2011

#### 7.11.2 Variable Documentation

##### 7.11.2.1 list forcebalance::custom_io::cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

Types of counterpoise correction.

Definition at line 16 of file custom_io.py.

##### 7.11.2.2 dictionary forcebalance::custom_io::fdict

**Initial value:**

```
1 {
2     'counterpoise'  : cptypes    }
```

Section -> Interaction type dictionary.

Definition at line 21 of file custom_io.py.

##### 7.11.2.3 list forcebalance::custom_io::ndtypes = [None]

Types of NDDO correction.

Definition at line 18 of file custom_io.py.

##### 7.11.2.4 dictionary forcebalance::custom_io::pdict

**Initial value:**

```
1 {'CPGAUSS':{3:'A', 4:'B', 5:'C'},
2          'CPGEXP' :{3:'A', 4:'B', 5:'G', 6:'X'},
3          'CPEXPG' :{3:'A1', 4:'B', 5:'X0', 6:'A2'}
4          }
```

Interaction type -> Parameter Dictionary.

Definition at line 25 of file custom_io.py.

### 7.12 forcebalance::finite_difference Namespace Reference

**Functions**

- def f1d2p

  *A two-point finite difference stencil.*
- def f1d5p

  *A highly accurate five-point finite difference stencil for computing derivatives of a function.*
- def f1d7p

  *A highly accurate seven-point finite difference stencil for computing derivatives of a function.*
- def f12d7p
- def f12d3p

  *A three-point finite difference stencil.*
- def in_fd

*Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.*

- def fdwrap

  *A function wrapper for finite difference designed for differentiating 'get'-type functions.*

- def fdwrap_G

  *A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.*

- def fdwrap_H

  *A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.*

### 7.12.1 Function Documentation

#### 7.12.1.1 def forcebalance.finite_difference.f12d3p ( *f, h, f0* = None )

A three-point finite difference stencil.

This function does either two computations or three, depending on whether the 'center' value is supplied. This is done in order to avoid recomputing the center value many times.

The first derivative is evaluated using central difference. One advantage of using central difference (as opposed to forward difference) is that we get zero at the bottom of a parabola.

Using this formula we also get an approximate second derivative, which can then be inserted into the diagonal of the Hessian. This is very useful for optimizations like BFGS where the diagonal determines how far we step in the parameter space.

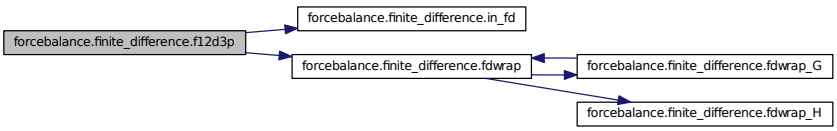
How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 107 of file finite_difference.py.

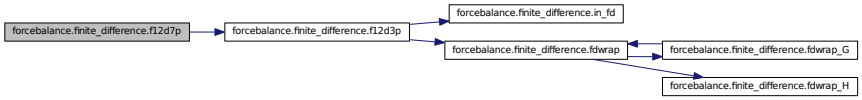
Here is the call graph for this function:




#### 7.12.1.2 def forcebalance.finite_difference.f12d7p ( *f, h* )

Definition at line 73 of file finite_difference.py.

Here is the call graph for this function:




#### 7.12.1.3 def forcebalance.finite_difference.f1d2p ( *f, h, f0* = None )

A two-point finite difference stencil.

This function does either two computations or one, depending on whether the 'center' value is supplied. This is done in order to avoid recomputing the center value many times when we repeat this function for each index of the gradient.

How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 27 of file finite_difference.py.

Here is the call graph for this function:

#### 7.12.1.4 def forcebalance.finite_difference.f1d5p ( *f, h* )

A highly accurate five-point finite difference stencil for computing derivatives of a function.

It works on both scalar and vector functions (i.e. functions that return arrays). Since the function does four computations, it's costly but recommended if we really need an accurate reference value.

The function is evaluated at points -2h, -h, +h and +2h and these values are combined to make the derivative according to: http://www.holoborodko.com/pavel/numerical-methods/numerical-derivative/central-differen

How to use: use fdwrap or something similar to generate a one-variable function from the (usually) much more complicated function that we wish to differentate. Then pass it to this function.

Inputs: f = The one-variable function f(x) that we're differentiating h = The finite difference step size, usually a small number

Outputs: fp = The finite difference derivative of the function f(x) around x=0.

Definition at line 58 of file finite_difference.py.

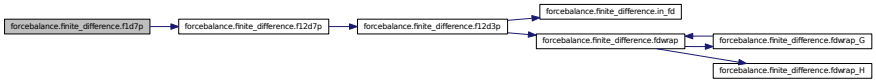
Here is the call graph for this function:

#### 7.12.1.5 def forcebalance.finite_difference.f1d7p ( *f, h* )

A highly accurate seven-point finite difference stencil for computing derivatives of a function.

Definition at line 68 of file finite_difference.py.

Here is the call graph for this function:




#### 7.12.1.6 def forcebalance.finite_difference.fdwrap ( *func, mvals0, pidx, key =* None*, kwargs* )

A function wrapper for finite difference designed for differentiating 'get'-type functions.

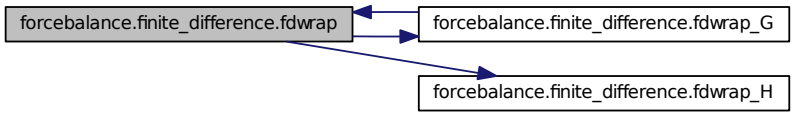
Since our finite difference stencils take single-variable functions and differentiate them around zero, and our objective function is quite a complicated function, we need a wrapper to serve as a middleman. The alternative would be to copy the finite difference formula to wherever we're taking the derivative, and that is prone to mistakes.

Inputs: func = Either get_X or get_G; these functions return dictionaries. ['X'] = 1.23, ['G'] = [0.12, 3,45, ...] mvals0 = The 'central' values of the mathematical parameters - i.e. the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating key = either 'G' or 'X', the value we wish to take out of the dictionary kwargs = Anything else we want to pass to the objective function (for instance, Project.Objective takes Order as an argument)

Outputs: func1 = Wrapped version of func, which takes a single float argument.

Definition at line 145 of file finite_difference.py.

Here is the call graph for this function:



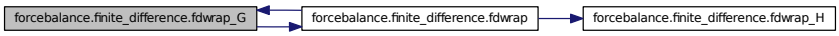

#### 7.12.1.7 def forcebalance.finite_difference.fdwrap_G ( *tgt, mvals0, pidx* )

A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating

Definition at line 164 of file finite_difference.py.

Here is the call graph for this function:




#### 7.12.1.8 def forcebalance.finite_difference.fdwrap_H ( *tgt, mvals0, pidx* )

A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.

the wrapped function's origin is here. pidx = The index of the parameter that we're differentiating

Definition at line 175 of file finite_difference.py.

#### 7.12.1.9 def forcebalance.finite_difference.in_fd ( )

Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.

This is mainly useful for deciding when to update the 'qualitative indicators' and when not to.

Definition at line 119 of file finite_difference.py.

## 7.13 forcebalance::forcefield Namespace Reference

Force field module.

### Classes

- class BackedUpDict
- class FF

  *Force field class.*

### Functions

- def determine_fftype

  *Determine the type of a force field file.*
- def rs_override

  *This function takes in a dictionary (rsfactors) and a string (termtype).*

### Variables

- dictionary FF_Extensions
- dictionary FF_IOModules

### 7.13.1 Detailed Description

Force field module. In ForceBalance a 'force field' is built from a set of files containing physical parameters. These files can be anything that enter into any computation - our original program was quite dependent on the GROMACS force field format, but this program is set up to allow very general input formats.

We introduce several important concepts:

1) Adjustable parameters are allocated into a vector.

To cast the force field optimization as a math problem, we treat all of the parameters on equal footing and write them as indices in a parameter vector.

2) A mapping from interaction type to parameter number.

Each element in the parameter vector corresponds to one or more interaction types. Whenever we change the parameter vector and recompute the objective function, this amounts to changing the physical parameters in the simulations, so we print out new force field files for external programs. In addition, when these programs are computing the objective function we are often in low-level subroutines that compute terms in the energy and force. If we need an analytic derivative of the objective function, then these subroutines need to know which index of the parameter vector needs to be modified.

This is done by way of a hash table: for example, when we are computing a Coulomb interaction between atom 4 and atom 5, we can build the words 'COUL4' and 'COUL5' and look it up in the parameter map; this gives us two numbers (say, 10 and 11) corresponding to the eleventh and twelfth element of the parameter vector. Then we can compute the derivatives of the energy w/r.t. these parameters (in this case, COUL5/rij and COUL4/rij) and increment these values in the objective function gradient.

In custom-implemented force fields (see counterpoisematch.py) the hash table can also be used to look up parameter values for computation of interactions. This is probably not the fastest way to do things, however.

3) Distinction between physical and mathematical parameters.

The optimization algorithm works in a space that is related to, but not exactly the same as the physical parameter space. The reasons for why we do this are:

a) Each parameter has its own physical units. On the one hand it's not right to treat different physical units all on the same footing, so nondimensionalization is desirable. To make matters worse, the force field parameters can be small as 1e-8 or as large as 1e+6 depending on the parameter type. This means the elements of the objective function gradient / Hessian have elements that differ from each other in size by 10+ orders of magnitude, leading to mathematical instabilities in the optimizer.

b) The parameter space can be constrained, most notably for atomic partial charges where we don't want to change the overall charge on a molecule. Thus we wish to project out certain movements in the mathematical parameters such that they don't change the physical parameters.

c) We wish to regularize our optimization so as to avoid changing our parameters in very insensitive directions (linear dependencies). However, the sensitivity of the objective function to changes in the force field depends on the physical units!

For all of these reasons, we introduce a 'transformation matrix' which maps mathematical parameters onto physical parameters. The diagonal elements in this matrix are rescaling factors; they take the mathematical parameter and magnify it by this constant factor. The off-diagonal elements correspond to rotations and other linear transformations, and currently I just use them to project out the 'increase the net charge' direction in the physical parameter space.

Note that with regularization, these rescaling factors are equivalent to the widths of prior distributions in a maximum likelihood framework. Because there is such a correspondence between rescaling factors and choosing a prior, they need to be chosen carefully. This is work in progress. Another possibility is to sample the width of the priors from a noninformative distribution -- the hyperprior (we can choose the Jeffreys prior or something). This is work in progress.

Right now only GROMACS parameters are supported, but this class is extensible, we need more modules!

**Author**

Lee-Ping Wang

**Date**

04/2012

### 7.13.2 Function Documentation

#### 7.13.2.1 def forcebalance.forcefield.determine_fftype ( *ffname, verbose* = `False` )

Determine the type of a force field file.

It is possible to specify the file type explicitly in the input file using the syntax 'force_field.ext:type'. Otherwise this function will try to determine the force field type by extension.

Definition at line 143 of file forcefield.py.

#### 7.13.2.2 def forcebalance.forcefield.rs_override ( *rsfactors, termtype, Temperature* = `298.15` )

This function takes in a dictionary (rsfactors) and a string (termtype).

If termtype matches any of the strings below, rsfactors[termtype] is assigned to one of the numbers below.

This is LPW's attempt to simplify the rescaling factors.

**Parameters**

| | | |
|---|---|---|
| `out` | *rsfactors* | The computed rescaling factor. |
| `in` | *termtype* | The interaction type (corresponding to a physical unit) |
| `in` | *Temperature* | The temperature for computing the kT energy scale |

Definition at line 1171 of file forcefield.py.

### 7.13.3 Variable Documentation

#### 7.13.3.1 dictionary forcebalance::forcefield::FF_Extensions

**Initial value:**

```
1 {"itp" : "gmx",
2                  "in"  : "qchem",
3                  "prm" : "tinker",
4                  "gen" : "custom",
5                  "xml" : "openmm",
6                  "frcmod" : "frcmod",
7                  "mol2" : "mol2",
8                  "gbs"  : "gbs",
9                  "grid" : "grid"
10                  }
```

Definition at line 115 of file forcefield.py.

#### 7.13.3.2 dictionary forcebalance::forcefield::FF_IOModules

**Initial value:**

```
1 {"gmx": gmxio.ITP_Reader ,
2                 "qchem": qchemio.QCIn_Reader ,
3                 "tinker": tinkerio.Tinker_Reader ,
4                 "custom": custom_io.Gen_Reader ,
5                 "openmm" : openmmio.OpenMM_Reader,
6                 "frcmod" : amberio.FrcMod_Reader,
```

```
7                 "mol2" : amberio.Mol2_Reader,
8                 "gbs" : psi4io.GBS_Reader,
9                 "grid" : psi4io.Grid_Reader
10                 }
```

Definition at line 127 of file forcefield.py.

## 7.14 forcebalance::gmxio Namespace Reference

GROMACS input/output.

### Classes

- class ITP_Reader

  *Finite state machine for parsing GROMACS force field files.*
- class AbInitio_GMX

  *Subclass of AbInitio for force and energy matching using normal GROMACS.*
- class Liquid_GMX
- class Interaction_GMX

  *Subclass of Interaction for interaction energy matching using GROMACS.*

### Functions

- def edit_mdp

  *Create or edit a Gromacs MDP file.*
- def parse_atomtype_line

  *Parses the 'atomtype' line.*
- def rm_gmx_baks

### Variables

- list nftypes = [None, 'VDW', 'VDW_BHAM']

  *VdW interaction function types.*
- list pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

  *Pairwise interaction function types.*
- list bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

  *Bonded interaction function types.*
- list aftypes

  *Angle interaction function types.*
- list dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

  *Dihedral interaction function types.*
- dictionary fdict

  *Section -> Interaction type dictionary.*
- dictionary pdict

  *Interaction type -> Parameter Dictionary.*

### 7.14.1 Detailed Description

GROMACS input/output.

**Todo** Even more stuff from forcefield.py needs to go into here.

**Author**

Lee-Ping Wang

**Date**

12/2011

**Todo** Even more stuff from forcefield.py needs to go into here.

**Author**

Lee-Ping Wang

**Date**

12/2011

### 7.14.2 Function Documentation

#### 7.14.2.1 def forcebalance.gmxio.edit_mdp ( *fin, fout, options, verbose* = `False` )

Create or edit a Gromacs MDP file.

**Parameters**

| | | |
|---|---|---|
| `in` | *fin* | Input file name. |
| `in` | *fout* | Output file name, can be the same as input file name. |
| `in` | *options* | Dictionary containing mdp options. Existing options are replaced, new options are added at the end. |

Definition at line 32 of file gmxio.py.

Here is the call graph for this function:


#### 7.14.2.2 def forcebalance.gmxio.parse_atomtype_line ( *line* )

Parses the 'atomtype' line.

Parses lines like this:

```
opls_135 CT 6 12.0107 0.0000 A 3.5000e-01 2.7614e-01
```

```
C 12.0107 0.0000 A 3.7500e-01 4.3932e-01

Na 11 22.9897 0.0000 A 6.068128070229e+03 2.662662556402e+01 0.0000e+00 ; PARM
5 6
```

Look at all the variety!

**Parameters**

| | | |
|---|---|---|
| in | *line* | Input line. |

**Returns**

answer Dictionary containing:
atom type
bonded atom type (if any)
atomic number (if any)
atomic mass
charge
particle type
force field parameters
number of optional fields

Definition at line 178 of file gmxio.py.

Here is the call graph for this function:


#### 7.14.2.3 def forcebalance.gmxio.rm_gmx_baks ( *dir* )

Definition at line 425 of file gmxio.py.

### 7.14.3 Variable Documentation

#### 7.14.3.1 list forcebalance::gmxio::aftypes

**Initial value:**

```
1 [None, 'ANGLES', 'G96ANGLES', 'CROSS_BOND_BOND',
2            'CROSS_BOND_ANGLE', 'UREY_BRADLEY', 'QANGLES']
```

Angle interaction function types.

Definition at line 88 of file gmxio.py.

#### 7.14.3.2 list forcebalance::gmxio::bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

Bonded interaction function types.

Definition at line 86 of file gmxio.py.

#### 7.14.3.3 list forcebalance::gmxio::dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

Dihedral interaction function types.

Definition at line 91 of file gmxio.py.

#### 7.14.3.4 dictionary forcebalance::gmxio::fdict

**Initial value:**

```
1 {
2     'atomtypes'     : nftypes,
3     'nonbond_params': pftypes,
4     'bonds'         : bftypes,
5     'bondtypes'     : bftypes,
6     'angles'        : aftypes,
7     'angletypes'    : aftypes,
8     'dihedrals'     : dftypes,
9     'dihedraltypes' : dftypes,
10     'virtual_sites2': ['NONE','VSITE2'],
11     'virtual_sites3': ['NONE','VSITE3','VSITE3FD','VSITE3FAD','VSITE3OUT'],
12     'virtual_sites4': ['NONE','VSITE4FD','VSITE4FDN']
13     }
```

Section -> Interaction type dictionary.

Based on the section you're in and the integer given on the current line, this looks up the 'interaction type' - for example, within bonded interactions there are four interaction types: harmonic, G96, Morse, and quartic interactions.

Definition at line 99 of file gmxio.py.

#### 7.14.3.5 list forcebalance::gmxio::nftypes = [None, 'VDW', 'VDW_BHAM']

VdW interaction function types.

Definition at line 82 of file gmxio.py.

#### 7.14.3.6 dictionary forcebalance::gmxio::pdict

Interaction type -> Parameter Dictionary.

A list of supported GROMACS interaction types in force matching. The keys in this dictionary (e.g. 'BONDS','ANGLES') are values in the interaction type dictionary. As the program loops through the force field file, it first looks up the interaction types in 'fdict' and then goes here to do the parameter lookup by field.

**Todo** This needs to become more flexible because the parameter isn't always in the same field. Still need to figure out how to do this.

How about making the PDIHS less ugly?

Definition at line 122 of file gmxio.py.

#### 7.14.3.7 list forcebalance::gmxio::pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

Pairwise interaction function types.

Definition at line 84 of file gmxio.py.

## 7.15 forcebalance::gmxqpio Namespace Reference

#### Classes

- class Monomer_QTPIE
  *Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).*

#### Functions

- def get_monomer_properties

### 7.15.1 Function Documentation

#### 7.15.1.1 def forcebalance.gmxqpio.get_monomer_properties ( *print_stuff* = 0 )

Definition at line 25 of file gmxqpio.py.

Here is the call graph for this function:


## 7.16 forcebalance::interaction Namespace Reference

Interaction energy fitting module.

#### Classes

- class Interaction
  *Subclass of Target for fitting force fields to interaction energies.*

### 7.16.1 Detailed Description

Interaction energy fitting module.

**Author**

Lee-Ping Wang

**Date**

05/2012

## 7.17 forcebalance::leastsq Namespace Reference

#### Classes

- class LeastSquares
  *Subclass of Target for general least squares fitting.*

#### Functions

- def CheckBasis
- def LastMvals

#### Variables

- CHECK_BASIS = False
- LAST_MVALS = None

### 7.17.1 Function Documentation

#### 7.17.1.1 def forcebalance.leastsq.CheckBasis ( )

Definition at line 21 of file leastsq.py.

#### 7.17.1.2 def forcebalance.leastsq.LastMvals ( )

Definition at line 26 of file leastsq.py.

### 7.17.2 Variable Documentation

#### 7.17.2.1 forcebalance::leastsq::CHECK_BASIS = False

Definition at line 20 of file leastsq.py.

#### 7.17.2.2 forcebalance::leastsq::LAST_MVALS = None

Definition at line 25 of file leastsq.py.

## 7.18 forcebalance::liquid Namespace Reference

Matching of liquid bulk properties.

#### Classes

- class Liquid

  *Subclass of Target for liquid property matching.*

#### Functions

- def weight_info

### 7.18.1 Detailed Description

Matching of liquid bulk properties. Under development.

**Author**

Lee-Ping Wang

**Date**

04/2012

#### 7.18.2 Function Documentation

##### 7.18.2.1 def forcebalance.liquid.weight_info ( *W, PT, N_k, verbose* = `True` )

Definition at line 29 of file liquid.py.

## 7.19 forcebalance::Mol2 Namespace Reference

**Classes**

- class mol2_atom

  *This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.*
- class mol2_bond

  *This is to manage mol2 bond lines on the form: 1 1 2 ar.*
- class mol2

  *This is to manage one mol2 series of lines on the form:*
- class mol2_set

**Variables**

- tuple data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

#### 7.19.1 Variable Documentation

##### 7.19.1.1 tuple forcebalance::Mol2::data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

Definition at line 651 of file Mol2.py.

## 7.20 forcebalance::mol2io Namespace Reference

Mol2 I/O.

**Classes**

- class Mol2_Reader

  *Finite state machine for parsing Mol2 force field file.*

**Variables**

- dictionary mol2_pdict = {'COUL':{'Atom':[1], 6:"}}

#### 7.20.1 Detailed Description

Mol2 I/O. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

**Author**

Lee-Ping Wang

**Date**

05/2012

#### 7.20.2 Variable Documentation

##### 7.20.2.1 dictionary forcebalance::mol2io::mol2_pdict = {'COUL':{'Atom':[1], 6:''}}

Definition at line 18 of file mol2io.py.

## 7.21 forcebalance::molecule Namespace Reference

**Classes**

- class MolfileTimestep

  *Wrapper for the timestep C structure used in molfile plugins.*
- class Molecule

  *Lee-Ping's general file format conversion class.*

**Functions**

- def getElement
- def nodematch
- def isint

  *ONLY matches integers! If you have a decimal point? None shall pass!*
- def isfloat

  *Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.*
- def BuildLatticeFromLengthsAngles

  *This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.*
- def BuildLatticeFromVectors

  *This function takes in three lattice vectors and tries to return a complete box specification.*
- def format_xyz_coord

  *Print a line consisting of (element, x, y, z) in accordance with .xyz file format.*
- def format_gro_coord

  *Print a line in accordance with .gro file format, with six decimal points of precision.*
- def format_xyzgen_coord

  *Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)*
- def format_gro_box

  *Print a line corresponding to the box vector in accordance with .gro file format.*

- def is_gro_coord

  *Determines whether a line contains GROMACS data or not.*
- def is_charmm_coord

  *Determines whether a line contains CHARMM data or not.*
- def is_gro_box

  *Determines whether a line contains a GROMACS box vector or not.*
- def add_strip_to_mat
- def pvec
- def grouper

  *Groups a big long iterable into groups of ten or what have you.*
- def even_list

  *Creates a list of number sequences divided as evenly as possible.*
- def both
- def diff
- def either
- def EulerMatrix

  *Constructs an Euler matrix from three Euler angles.*
- def ComputeOverlap

  *Computes an 'overlap' between two molecules based on some fictitious density.*
- def AlignToDensity

  *Computes a "overlap density" from two frames.*
- def AlignToMoments

  *Pre-aligns molecules to 'moment of inertia'.*
- def get_rotate_translate
- def main

**Variables**

- tuple FrameVariableNames
- tuple AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])
- tuple MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])
- tuple QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])
- AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames
- list Radii
- list Elements
- tuple PeriodicTable
- float bohrang = 0.529177249

  *One bohr equals this many angstroms.*
- tuple splitter = re.compile(r'(\s+|\S+)')
- tuple Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])
- int radian = 180
- Alive

### 7.21.1 Function Documentation

#### 7.21.1.1 def forcebalance.molecule.add_strip_to_mat ( *mat, strip* )

Definition at line 438 of file molecule.py.

Here is the call graph for this function:

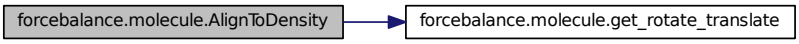

#### 7.21.1.2 def forcebalance.molecule.AlignToDensity ( *elem, xyz1, xyz2, binary* = `False` )

Computes a "overlap density" from two frames.

This function can be called by AlignToMoments to get rid of inversion problems

Definition at line 541 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.3 def forcebalance.molecule.AlignToMoments ( *elem, xyz1, xyz2* = `None` )

Pre-aligns molecules to 'moment of inertia'.

If xyz2 is passed in, it will assume that xyz1 is already aligned to the moment of inertia, and it simply does 180-degree rotations to make sure nothing is inverted.

Definition at line 553 of file molecule.py.

#### 7.21.1.4 def forcebalance.molecule.both ( *A, B, key* )

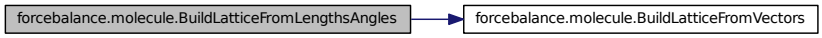
Definition at line 476 of file molecule.py.

#### 7.21.1.5 def forcebalance.molecule.BuildLatticeFromLengthsAngles ( *a, b, c, alpha, beta, gamma* )

This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.

Definition at line 258 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.6 def forcebalance.molecule.BuildLatticeFromVectors ( *v1, v2, v3* )

This function takes in three lattice vectors and tries to return a complete box specification.

The hash function is something we can use to discard two things that are obviously not equal. Here we neglect the hash. Return a list of the sorted atom numbers in this graph. Return a string of atoms, which serves as a rudimentary 'fingerprint' : '99,100,103,151' . Return an array of the elements. For instance ['H' 'C' 'C' 'H']. Create an Empirical Formula Get a list of the coordinates.

Definition at line 273 of file molecule.py.

#### 7.21.1.7 def forcebalance.molecule.ComputeOverlap ( *theta, elem, xyz1, xyz2* )

Computes an 'overlap' between two molecules based on some fictitious density.

Good for fine-tuning alignment but gets stuck in local minima.

Definition at line 524 of file molecule.py.

Here is the call graph for this function:


#### 7.21.1.8 def forcebalance.molecule.diff ( *A, B, key* )

Definition at line 479 of file molecule.py.

#### 7.21.1.9 def forcebalance.molecule.either ( *A, B, key* )

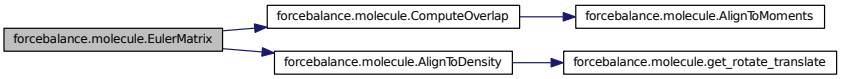
Definition at line 487 of file molecule.py.

#### 7.21.1.10 def forcebalance.molecule.EulerMatrix ( *T1, T2, T3* )

Constructs an Euler matrix from three Euler angles.

Definition at line 496 of file molecule.py.

Here is the call graph for this function:

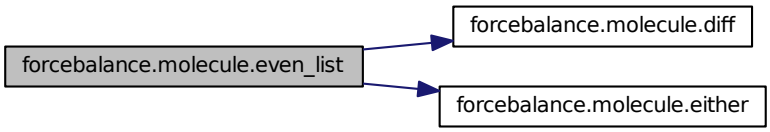



#### 7.21.1.11 def forcebalance.molecule.even_list ( *totlen, splitsize* )

Creates a list of number sequences divided as evenly as possible.

Definition at line 458 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.12 def forcebalance.molecule.format_gro_box ( *box* )

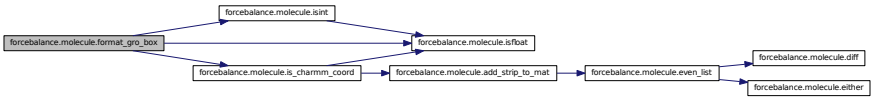
Print a line corresponding to the box vector in accordance with .gro file format.

**Parameters**

| | | |
|---|---|---|
| in | *box* | Box NamedTuple |

Definition at line 389 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.13 def forcebalance.molecule.format_gro_coord ( *resid, resname, aname, seqno, xyz* )

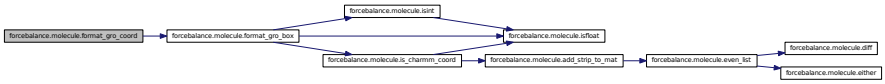
Print a line in accordance with .gro file format, with six decimal points of precision.

**Parameters**

| | | |
|---|---|---|
| in | *resid* | The number of the residue that the atom belongs to |
| in | *resname* | The name of the residue that the atom belongs to |
| in | *aname* | The name of the atom |
| in | *seqno* | The sequential number of the atom |
| in | *xyz* | A 3-element array containing x, y, z coordinates of that atom |

Definition at line 368 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.14 def forcebalance.molecule.format_xyz_coord ( *element, xyz, tinker* = `False` )

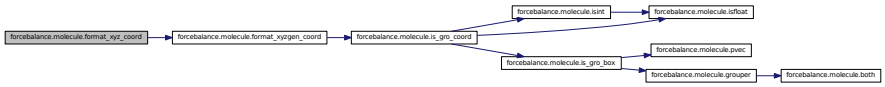
Print a line consisting of (element, x, y, z) in accordance with .xyz file format.

**Parameters**

| | | |
|---|---|---|
| in | *element* | A chemical element of a single atom |
| in | *xyz* | A 3-element array containing x, y, z coordinates of that atom |

Definition at line 352 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.15 def forcebalance.molecule.format_xyzgen_coord ( *element, xyzgen* )

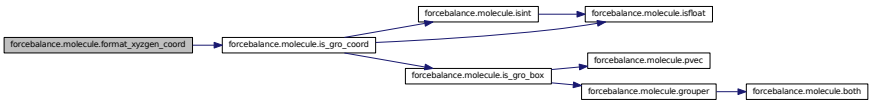
Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)

**Parameters**

| | | |
|---|---|---|
| in | *element* | A chemical element of a single atom |
| in | *xyzgen* | A N-element array containing data for that atom |

Definition at line 380 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.16 def forcebalance.molecule.get_rotate_translate ( *matrix1, matrix2* )

Definition at line 576 of file molecule.py.

#### 7.21.1.17 def forcebalance.molecule.getElement ( *mass* )

Definition at line 191 of file molecule.py.

#### 7.21.1.18 def forcebalance.molecule.grouper ( *n, iterable* )

Groups a big long iterable into groups of ten or what have you.

Definition at line 452 of file molecule.py.

Here is the call graph for this function:


#### 7.21.1.19 def forcebalance.molecule.is_charmm_coord ( *line* )

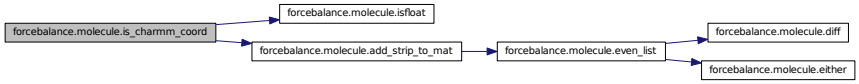
Determines whether a line contains CHARMM data or not.

**Parameters**

| | | |
|---|---|---|
| in | *line* | The line to be tested |

Definition at line 416 of file molecule.py.

Here is the call graph for this function:



#### 7.21.1.20 def forcebalance.molecule.is_gro_box ( *line* )

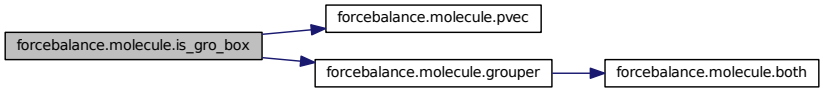
Determines whether a line contains a GROMACS box vector or not.

**Parameters**

| | | |
|---|---|---|
| in | *line* | The line to be tested |

Definition at line 429 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.21 def forcebalance.molecule.is_gro_coord ( *line* )

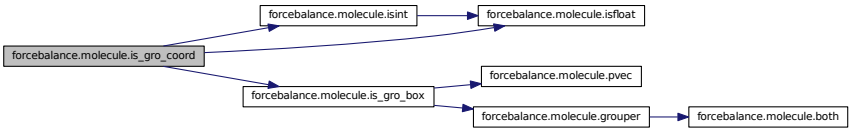
Determines whether a line contains GROMACS data or not.

**Parameters**

| | | |
|---|---|---|
| in | *line* | The line to be tested |

Definition at line 401 of file molecule.py.

Here is the call graph for this function:




#### 7.21.1.22 def forcebalance.molecule.isfloat ( *word* )

Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.

Definition at line 247 of file molecule.py.

#### 7.21.1.23 def forcebalance.molecule.isint ( *word* )

ONLY matches integers! If you have a decimal point? None shall pass!

Definition at line 242 of file molecule.py.

Here is the call graph for this function:


#### 7.21.1.24 def forcebalance.molecule.main ( )

Definition at line 2489 of file molecule.py.

#### 7.21.1.25 def forcebalance.molecule.nodematch ( *node1, node2* )

Definition at line 236 of file molecule.py.

Here is the call graph for this function:


#### 7.21.1.26 def forcebalance.molecule.pvec ( *vec* )

Definition at line 447 of file molecule.py.

### 7.21.2 Variable Documentation

#### 7.21.2.1 forcebalance::molecule::Alive

Definition at line 278 of file molecule.py.

#### 7.21.2.2 forcebalance::molecule::AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames

Definition at line 139 of file molecule.py.

#### 7.21.2.3 tuple forcebalance::molecule::AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])

Definition at line 122 of file molecule.py.

#### 7.21.2.4 float forcebalance::molecule::bohrang = 0.529177249

One bohr equals this many angstroms.

Definition at line 234 of file molecule.py.

#### 7.21.2.5 tuple forcebalance::molecule::Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])

Definition at line 254 of file molecule.py.

#### 7.21.2.6 list forcebalance::molecule::Elements

**Initial value:**

```
1 ["None",'H','He',
2             'Li','Be','B','C','N','O','F','Ne',
3             'Na','Mg','Al','Si','P','S','Cl','Ar',
4             'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge'
      ,'As','Se','Br','Kr',
5             'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn
      ','Sb','Te','I','Xe',
6             'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','
      Er','Tm','Yb',
7             'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po
      ','At','Rn',
8             'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm
      ','Md','No','Lr','Rf','Db','Sg','Bh','Hs','Mt']
```

Definition at line 166 of file molecule.py.

#### 7.21.2.7 tuple forcebalance::molecule::FrameVariableNames

**Initial value:**

```
1 set(['xyzs', 'comms', 'boxes', 'qm_forces', 'qm_energies', 'qm_interaction',
2                          'qm_espxyzs', 'qm_espvals', 'qm_extchgs', '
      qm_mulliken_charges', 'qm_mulliken_spins'])
```

Definition at line 108 of file molecule.py.

#### 7.21.2.8 tuple forcebalance::molecule::MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])

Definition at line 135 of file molecule.py.

#### 7.21.2.9 tuple forcebalance::molecule::PeriodicTable

**Initial value:**

```
1 OrderedDict([('H' , 1.0079), ('He' , 4.0026),
2                          ('Li' , 6.941), ('Be' , 9.0122), ('B' , 10.811), (
      'C' , 12.0107), ('N' , 14.0067), ('O' , 15.9994), ('F' , 18.9984), ('Ne' , 20.17
      97),
3                          ('Na' , 22.9897), ('Mg' , 24.305), ('Al' , 26.9815
      ), ('Si' , 28.0855), ('P' , 30.9738), ('S' , 32.065), ('Cl' , 35.453), ('Ar' , 3
      9.948),
4                          ('K' , 39.0983), ('Ca' , 40.078), ('Sc' , 44.9559)
      , ('Ti' , 47.867), ('V' , 50.9415), ('Cr' , 51.9961), ('Mn' , 54.938), ('Fe' , 5
      5.845), ('Co' , 58.9332),
5                          ('Ni' , 58.6934), ('Cu' , 63.546), ('Zn' , 65.39),
       ('Ga' , 69.723), ('Ge' , 72.64), ('As' , 74.9216), ('Se' , 78.96), ('Br' , 79.9
      04), ('Kr' , 83.8),
6                          ('Rb' , 85.4678), ('Sr' , 87.62), ('Y' , 88.9059),
       ('Zr' , 91.224), ('Nb' , 92.9064), ('Mo' , 95.94), ('Tc' , 98), ('Ru' , 101.07)
      , ('Rh' , 102.9055),
7                          ('Pd' , 106.42), ('Ag' , 107.8682), ('Cd' , 112.41
      1), ('In' , 114.818), ('Sn' , 118.71), ('Sb' , 121.76), ('Te' , 127.6), ('I' , 1
      26.9045), ('Xe' , 131.293),
8                          ('Cs' , 132.9055), ('Ba' , 137.327), ('La' , 138.9
```

```
      055), ('Ce' , 140.116), ('Pr' , 140.9077), ('Nd' , 144.24), ('Pm' , 145), ('Sm'
      , 150.36),
9                             ('Eu' , 151.964), ('Gd' , 157.25), ('Tb' , 158.925
      3), ('Dy' , 162.5), ('Ho' , 164.9303), ('Er' , 167.259), ('Tm' , 168.9342), ('Yb
      ' , 173.04),
10                            ('Lu' , 174.967), ('Hf' , 178.49), ('Ta' , 180.947
      9), ('W' , 183.84), ('Re' , 186.207), ('Os' , 190.23), ('Ir' , 192.217), ('Pt' ,
       195.078),
11                            ('Au' , 196.9665), ('Hg' , 200.59), ('Tl' , 204.38
      33), ('Pb' , 207.2), ('Bi' , 208.9804), ('Po' , 209), ('At' , 210), ('Rn' , 222)
      ,
12                            ('Fr' , 223), ('Ra' , 226), ('Ac' , 227), ('Th' ,
      232.0381), ('Pa' , 231.0359), ('U' , 238.0289), ('Np' , 237), ('Pu' , 244),
13                            ('Am' , 243), ('Cm' , 247), ('Bk' , 247), ('Cf' ,
      251), ('Es' , 252), ('Fm' , 257), ('Md' , 258), ('No' , 259),
14                            ('Lr' , 262), ('Rf' , 261), ('Db' , 262), ('Sg' ,
      266), ('Bh' , 264), ('Hs' , 277), ('Mt' , 268)])
```

Definition at line 176 of file molecule.py.

#### 7.21.2.10 tuple forcebalance::molecule::QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])

Definition at line 137 of file molecule.py.

#### 7.21.2.11 int forcebalance::molecule::radian = 180

Definition at line 255 of file molecule.py.

#### 7.21.2.12 list forcebalance::molecule::Radii

**Initial value:**

```
1 [0.31, 0.28, # H and He
2          1.28, 0.96, 0.84, 0.76, 0.71, 0.66, 0.57, 0.58, # First row elements
3          1.66, 1.41, 1.21, 1.11, 1.07, 1.05, 1.02, 1.06, # Second row elements
4          2.03, 1.76, 1.70, 1.60, 1.53, 1.39, 1.61, 1.52, 1.50,
5          1.24, 1.32, 1.22, 1.22, 1.20, 1.19, 1.20, 1.20, 1.16, # Third row
       elements, K through Kr
6          2.20, 1.95, 1.90, 1.75, 1.64, 1.54, 1.47, 1.46, 1.42,
7          1.39, 1.45, 1.44, 1.42, 1.39, 1.39, 1.38, 1.39, 1.40, # Fourth row
       elements, Rb through Xe
8          2.44, 2.15, 2.07, 2.04, 2.03, 2.01, 1.99, 1.98,
9          1.98, 1.96, 1.94, 1.92, 1.92, 1.89, 1.90, 1.87, # Fifth row elements,
       s and f blocks
10          1.87, 1.75, 1.70, 1.62, 1.51, 1.44, 1.41, 1.36,
11          1.36, 1.32, 1.45, 1.46, 1.48, 1.40, 1.50, 1.50, # Fifth row elements,
       d and p blocks
12          2.60, 2.21, 2.15, 2.06, 2.00, 1.96, 1.90, 1.87, 1.80, 1.69]
```

Definition at line 152 of file molecule.py.

#### 7.21.2.13 tuple forcebalance::molecule::splitter = re.compile(r'(\s+|\S+)')

Definition at line 251 of file molecule.py.

## 7.22 forcebalance::moments Namespace Reference

Multipole moment fitting module.

#### Classes

- class Moments

  *Subclass of Target for fitting force fields to multipole moments (from experiment or theory).*

### 7.22.1 Detailed Description

Multipole moment fitting module.

#### Author

Lee-Ping Wang

#### Date

09/2012

## 7.23 forcebalance::nifty Namespace Reference

Nifty functions, intended to be imported by any module within ForceBalance.

#### Classes

- class Pickler_LP

  *A subclass of the python Pickler that implements pickling of _ElementTree types.*
- class Unpickler_LP

  *A subclass of the python Unpickler that implements unpickling of _ElementTree types.*
- class RawStreamHandler

  *Exactly like logging.StreamHandler except it does no extra formatting before sending logging messages to the stream.*
- class RawFileHandler

  *Exactly like logging.FileHandler except it does no extra formatting before sending logging messages to the file.*

#### Functions

- def pvec1d

  *Printout of a 1-D vector.*
- def pmat2d

  *Printout of a 2-D matrix.*
- def encode
- def segments
- def commadash
- def uncommadash
- def printcool

  *Cool-looking printout for slick formatting of output.*
- def printcool_dictionary

  *See documentation for printcool; this is a nice way to print out keys/values in a dictionary.*
- def isint

  *ONLY matches integers! If you have a decimal point? None shall pass!*

- def isfloat

  *Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.*
- def isdecimal

  *Matches things with a decimal only; see isint and isfloat.*
- def floatornan

  *Returns a big number if we encounter NaN.*
- def col

  *Given any list, array, or matrix, return a 1-column matrix.*
- def row

  *Given any list, array, or matrix, return a 1-row matrix.*
- def flat

  *Given any list, array, or matrix, return a single-index array.*
- def orthogonalize

  *Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.*
- def invert_svd

  *Invert a matrix using singular value decomposition.*
- def get_least_squares
- def statisticalInefficiency

  *Compute the (cross) statistical inefficiency of (two) timeseries.*
- def lp_dump

  *Use this instead of pickle.dump for pickling anything that contains _ElementTree types.*
- def lp_load

  *Use this instead of pickle.load for unpickling anything that contains _ElementTree types.*
- def getWorkQueue
- def getWQIds
- def createWorkQueue
- def queue_up

  *Submit a job to the Work Queue.*
- def queue_up_src_dest

  *Submit a job to the Work Queue.*
- def wq_wait1

  *This function waits ten seconds to see if a task in the Work Queue has finished.*
- def wq_wait

  *This function waits until the work queue is completely empty.*
- def GoInto
- def allsplit
- def Leave
- def MissingFileInspection
- def LinkFile
- def CopyFile
- def link_dir_contents
- def remove_if_exists

  *Remove the file if it exists (doesn't return an error).*
- def which
- def warn_press_key
- def warn_once

  *Prints a warning but will only do so once in a given run.*
- def concurrent_map

*Similar to the bultin function map().*

- def multiopen

  *This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:*

**Variables**

- float kb = 0.0083144100163

  *Boltzmann constant.*
- float eqcgmx = 2625.5002

  *Q-Chem to GMX unit conversion for energy.*
- float fqcgmx = 49621.9

  *Q-Chem to GMX unit conversion for force.*
- float bohrang = 0.529177249

  *One bohr equals this many angstroms.*
- string XMLFILE = 'x'

  *Pickle uses 'flags' to pickle and unpickle different variable types.*
- WORK_QUEUE = None
- tuple WQIDS = defaultdict(list)
- list specific_lst
- tuple specific_dct = dict(list(itertools.chain(∗[[(j,i[1]) for j in i[0]] for i in specific_lst])))

### 7.23.1 Detailed Description

Nifty functions, intended to be imported by any module within ForceBalance. Table of Contents:

- I/O formatting
- Math: Variable manipulation, linear algebra, least squares polynomial fitting
- Pickle: Expand Python's own pickle to accommodate writing XML etree objects
- Commands for submitting things to the Work Queue
- Various file and process management functions
- Development stuff (not commonly used)

Named after the mighty Sniffy Handy Nifty (King Sniffy)

**Author**

Lee-Ping Wang

**Date**

12/2011

### 7.23.2 Function Documentation

#### 7.23.2.1 def forcebalance.nifty.allsplit ( *Dir* )

Definition at line 711 of file nifty.py.

Here is the call graph for this function:


#### 7.23.2.2 def forcebalance.nifty.col ( *vec* )

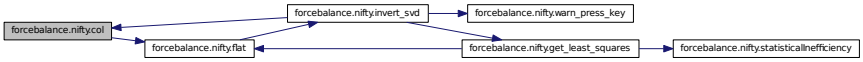
Given any list, array, or matrix, return a 1-column matrix.

Input: vec = The input vector that is to be made into a column

Output: A column matrix

Definition at line 267 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.3 def forcebalance.nifty.commadash ( *l* )

Definition at line 78 of file nifty.py.

Here is the call graph for this function:

#### 7.23.2.4 def forcebalance.nifty.concurrent_map ( *func, data* )

Similar to the bultin function map().

But spawn a thread for each argument and apply ‘func‘ concurrently.

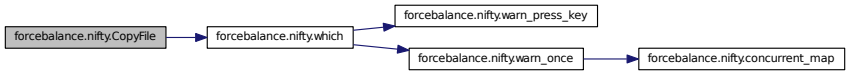
Note: unlike map(), we cannot take an iterable argument. ‘data‘ should be an indexable sequence.

Definition at line 942 of file nifty.py.

#### 7.23.2.5 def forcebalance.nifty.CopyFile ( *src, dest* )

Definition at line 759 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.6 def forcebalance.nifty.createWorkQueue ( *wq_port* )

Definition at line 576 of file nifty.py.

Here is the call graph for this function:


#### 7.23.2.7 def forcebalance.nifty.encode ( *l* )

Definition at line 69 of file nifty.py.

Here is the call graph for this function:

#### 7.23.2.8 def forcebalance.nifty.flat ( *vec* )

Given any list, array, or matrix, return a single-index array.

**Parameters**

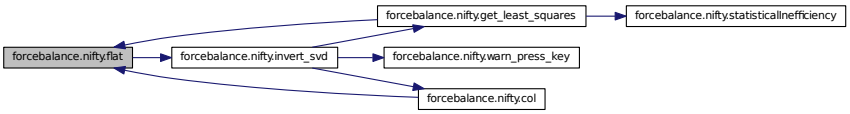
| | | |
|---|---|---|
| in | *vec* | The data to be flattened |

**Returns**

answer The flattened data

Definition at line 286 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.9 def forcebalance.nifty.floatornan ( *word* )

Returns a big number if we encounter NaN.

**Parameters**

| | | |
|---|---|---|
| in | *word* | The string to be converted |

**Returns**

answer The string converted to a float; if not a float, return 1e10

**Todo** I could use suggestions for making this better.

Definition at line 249 of file nifty.py.

Here is the call graph for this function:


#### 7.23.2.10 def forcebalance.nifty.get_least_squares ( *x*, *y*, *w* = None, *thresh* = 1e-12 )

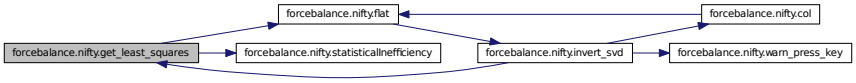
```
 __                  __
|                      |
| 1 (x0) (x0)^2 (x0)^3 |
| 1 (x1) (x1)^2 (x1)^3 |
| 1 (x2) (x2)^2 (x2)^3 |
| 1 (x3) (x3)^2 (x3)^3 |
| 1 (x4) (x4)^2 (x4)^3 |
|__                  __|
```

**Parameters**

| | | |
|---|---|---|
| in | *X* | (2-D array) An array of X-values (see above) |
| in | *Y* | (array) An array of Y-values (only used in getting the least squares coefficients) |
| in | *w* | (array) An array of weights, hopefully normalized to one. |
| out | *Beta* | The least-squares coefficients |
| out | *Hat* | The hat matrix that takes linear combinations of data y-values to give fitted y-values (weights) |
| out | *yfit* | The fitted y-values |
| out | *MPPI* | The Moore-Penrose pseudoinverse (multiply by Y to get least-squares coefficients, multiply by dY/dk to get derivatives of least-squares coefficients) |

Definition at line 354 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.11 def forcebalance.nifty.getWorkQueue ( )

Definition at line 568 of file nifty.py.

#### 7.23.2.12 def forcebalance.nifty.getWQIds ( )

Definition at line 572 of file nifty.py.

#### 7.23.2.13 def forcebalance.nifty.GoInto ( *Dir* )

Definition at line 703 of file nifty.py.

#### 7.23.2.14 def forcebalance.nifty.invert_svd ( *X*, *thresh* = 1e-12 )

Invert a matrix using singular value decomposition.

**Parameters**

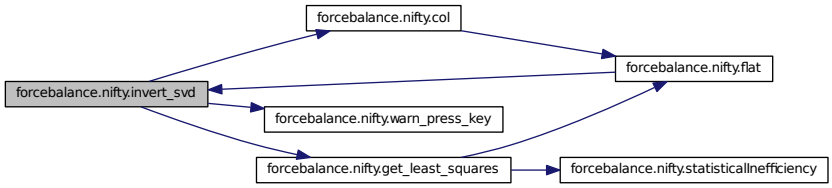
| | | |
|---|---|---|
| in | *X* | The matrix to be inverted |
| in | *thresh* | The SVD threshold; eigenvalues below this are not inverted but set to zero |

**Returns**

Xt The inverted matrix

Definition at line 313 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.15 def forcebalance.nifty.isdecimal ( *word* )

Matches things with a decimal only; see isint and isfloat.

**Parameters**

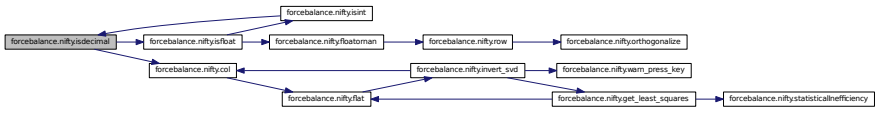
| | | |
|---|---|---|
| in | *word* | String (for instance, '123', '153.0', '2.', '-354') |

**Returns**

answer Boolean which specifies whether the string is a number with a decimal point

Definition at line 239 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.16 def forcebalance.nifty.isfloat ( *word* )

Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.

**Parameters**

| | | |
|---|---|---|
| in | *word* | String (for instance, '123', '153.0', '2.', '-354') |

**Returns**

answer Boolean which specifies whether the string is any number

Definition at line 229 of file nifty.py.

Here is the call graph for this function:


#### 7.23.2.17 def forcebalance.nifty.isint ( *word* )

ONLY matches integers! If you have a decimal point? None shall pass!

**Parameters**

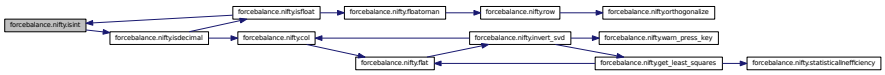
| | | |
|---|---|---|
| in | *word* | String (for instance, '123', '153.0', '2.', '-354') |

**Returns**

answer Boolean which specifies whether the string is an integer (only +/- sign followed by digits)

Definition at line 218 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.18 def forcebalance.nifty.Leave ( *Dir* )

Definition at line 717 of file nifty.py.

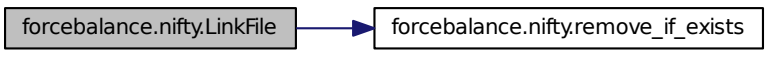
#### 7.23.2.19 def forcebalance.nifty.link_dir_contents ( *abssrcdir, absdestdir* )

Definition at line 769 of file nifty.py.

#### 7.23.2.20 def forcebalance.nifty.LinkFile ( *src, dest* )

Definition at line 748 of file nifty.py.

Here is the call graph for this function:

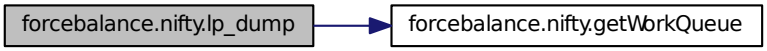


#### 7.23.2.21 def forcebalance.nifty.lp_dump ( *obj, file, protocol* = None )

Use this instead of pickle.dump for pickling anything that contains _ElementTree types.

Definition at line 546 of file nifty.py.

Here is the call graph for this function:

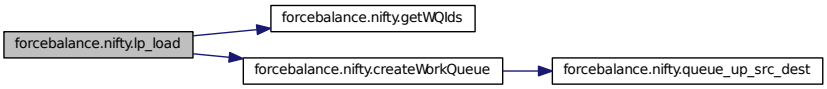



#### 7.23.2.22 def forcebalance.nifty.lp_load ( *file* )

Use this instead of pickle.load for unpickling anything that contains _ElementTree types.

Definition at line 551 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.23 def forcebalance.nifty.MissingFileInspection ( *fnm* )

Definition at line 738 of file nifty.py.

Here is the call graph for this function:


#### 7.23.2.24 def forcebalance.nifty.multiopen ( *arg* )

This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:

[file1, file2, file3 ... ]

each of which can then be iterated over:

[[file1_line1, file1_line2 ... ], [file2_line1, file2_line2 ... ]]

Definition at line 972 of file nifty.py.

#### 7.23.2.25 def forcebalance.nifty.orthogonalize ( *vec1, vec2* )

Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.

**Parameters**

| | | |
|---|---|---|
| in | *vec1* | The projectee (i.e. output is some modified version of vec1) |
| in | *vec2* | The projector (component subtracted out from vec1 is parallel to this) |

**Returns**

answer A copy of vec1 but with the vec2-component projected out.

Definition at line 300 of file nifty.py.

#### 7.23.2.26 def forcebalance.nifty.pmat2d ( *mat2d, precision* = 1 )

Printout of a 2-D matrix.

**Parameters**

| | | |
|---|---|---|
| in | *mat2d* | a 2-D matrix |

Definition at line 62 of file nifty.py.

Here is the call graph for this function:

#### 7.23.2.27 def forcebalance.nifty.printcool ( *text, sym* = "#", *bold* = False, *color* = 2, *ansi* = None, *bottom* = '-', *minwidth* = 50, *center* = True )

Cool-looking printout for slick formatting of output.

**Parameters**

| | | |
|---|---|---|
| in | *text* | The string that the printout is based upon. This function will print out the string, ANSI-colored and enclosed in the symbol for example:<br>`##################`<br>`### I am cool ###`<br>`##################` |
| in | *sym* | The surrounding symbol |
| in | *bold* | Whether to use bold print |
| in | *color* | The ANSI color:<br>1 red<br>2 green<br>3 yellow<br>4 blue<br>5 magenta<br>6 cyan<br>7 white |

| | | |
|---|---|---|
| in | *bottom* | The symbol for the bottom bar |
| in | *minwidth* | The minimum width for the box, if the text is very short then we insert the appropriate number of padding spaces |

**Returns**

bar The bottom bar is returned for the user to print later, e.g. to mark off a 'section'

Definition at line 151 of file nifty.py.

Here is the call graph for this function:


### 7.23.2.28 def forcebalance.nifty.printcool_dictionary ( *Dict, title* = "General options", *bold* = False, *color* = 2, *keywidth* = 25, *topwidth* = 50, *center* = True, *leftpad* = 0 )

See documentation for printcool; this is a nice way to print out keys/values in a dictionary.

The keys in the dictionary are sorted before printing out.

**Parameters**

| | | |
|---|---|---|
| in | *dict* | The dictionary to be printed |
| in | *title* | The title of the printout |

Definition at line 194 of file nifty.py.

Here is the call graph for this function:


### 7.23.2.29 def forcebalance.nifty.pvec1d ( *vec1d, precision* = 1 )

Printout of a 1-D vector.

**Parameters**

| | | |
|---|---|---|
| in | *vec1d* | a 1-D vector |

Definition at line 51 of file nifty.py.

Here is the call graph for this function:

#### 7.23.2.30 def forcebalance.nifty.queue_up ( *wq, command, input_files, output_files, tgt* = None, *verbose* = True )

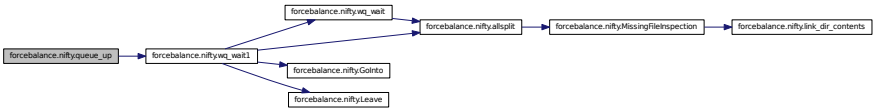
Submit a job to the Work Queue.

**Parameters**

| | | |
|---|---|---|
| in | *wq* | (Work Queue Object) |
| in | *command* | (string) The command to run on the remote worker. |
| in | *input_files* | (list of files) A list of locations of the input files. |
| in | *output_files* | (list of files) A list of locations of the output files. |

Definition at line 593 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.31 def forcebalance.nifty.queue_up_src_dest ( *wq, command, input_files, output_files, tgt* = None, *verbose* = True )

Submit a job to the Work Queue.

This function is a bit fancier in that we can explicitly specify where the input files come from, and where the output files go to.

**Parameters**

| | | |
|---|---|---|
| in | *wq* | (Work Queue Object) |
| in | *command* | (string) The command to run on the remote worker. |
| in | *input_files* | (list of 2-tuples) A list of local and remote locations of the input files. |
| in | *output_files* | (list of 2-tuples) A list of local and remote locations of the output files. |

Definition at line 625 of file nifty.py.

#### 7.23.2.32 def forcebalance.nifty.remove_if_exists ( *fnm* )

Remove the file if it exists (doesn't return an error).

Definition at line 780 of file nifty.py.

#### 7.23.2.33 def forcebalance.nifty.row ( *vec* )

Given any list, array, or matrix, return a 1-row matrix.

**Parameters**

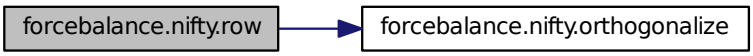
| | | |
|---|---|---|
| in | *vec* | The input vector that is to be made into a row |

**Returns**

answer A row matrix

Definition at line 277 of file nifty.py.

Here is the call graph for this function:




#### 7.23.2.34 def forcebalance.nifty.segments ( *e* )

Definition at line 72 of file nifty.py.

Here is the call graph for this function:

#### 7.23.2.35 def forcebalance.nifty.statisticalInefficiency ( *A_n*, *B_n* = None, *fast* = False, *mintime* = 3, *warn* = True )

Compute the (cross) statistical inefficiency of (two) timeseries.

Notes The same timeseries can be used for both A_n and B_n to get the autocorrelation statistical inefficiency. The fast method described in Ref [1] is used to compute g.

References [1] J. D. Chodera, W. C. Swope, J. W. Pitera, C. Seok, and K. A. Dill. Use of the weighted histogram analysis method for the analysis of simulated and parallel tempering simulations. JCTC 3(1):26-41, 2007.

Examples

Compute statistical inefficiency of timeseries data with known correlation time.

>>> import timeseries >>> A_n = timeseries.generateCorrelatedTimeseries(N=100000, tau=5.0) >>> g = statisticalInefficiency(A_n, fast=True)

**Parameters**

| | | |
|---|---|---|
| in | *A_n* | (required, numpy array) - A_n[n] is nth value of timeseries A. Length is deduced from vector. |
| in | *B_n* | (optional, numpy array) - B_n[n] is nth value of timeseries B. Length is deduced from vector. If supplied, the cross-correlation of timeseries A and B will be estimated instead of the autocorrelation of timeseries A. |
| in | *fast* | (optional, boolean) - if True, will use faster (but less accurate) method to estimate correlation time, described in Ref. [1] (default: False) |
| in | *mintime* | (optional, int) - minimum amount of correlation function to compute (default: 3) - The algorithm terminates after computing the correlation time out to mintime when the correlation function furst goes negative. Note that this time may need to be increased if there is a strong initial negative peak in the correlation function. |

**Returns**

g The estimated statistical inefficiency (equal to 1 + 2 tau, where tau is the correlation time). We enforce g >= 1.0.

Definition at line 430 of file nifty.py.

#### 7.23.2.36 def forcebalance.nifty.uncommadash ( *s* )

Definition at line 88 of file nifty.py.

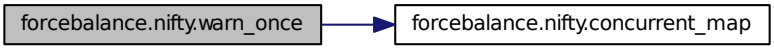
Here is the call graph for this function:

#### 7.23.2.37 def forcebalance.nifty.warn_once ( *warning*, *warnhash* = `None` )

Prints a warning but will only do so once in a given run.

Definition at line 890 of file nifty.py.

Here is the call graph for this function:



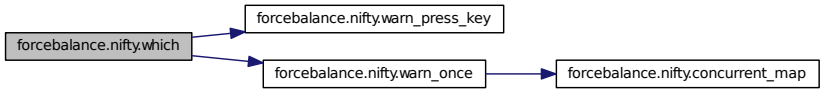

#### 7.23.2.38 def forcebalance.nifty.warn_press_key ( *warning*, *timeout* = `10` )

Definition at line 875 of file nifty.py.

#### 7.23.2.39 def forcebalance.nifty.which ( *fnm* )

Definition at line 784 of file nifty.py.

Here is the call graph for this function:

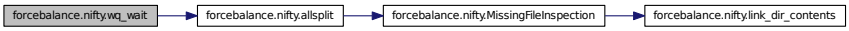



#### 7.23.2.40 def forcebalance.nifty.wq_wait ( *wq*, *verbose* = `False` )

This function waits until the work queue is completely empty.

Definition at line 696 of file nifty.py.

Here is the call graph for this function:

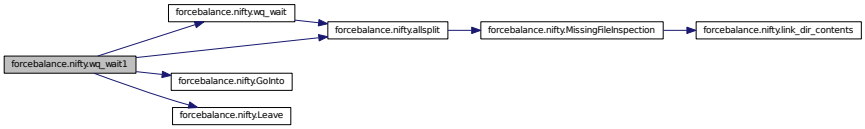



#### 7.23.2.41 def forcebalance.nifty.wq_wait1 ( *wq*, *wait_time* = `10`, *verbose* = `False` )

This function waits ten seconds to see if a task in the Work Queue has finished.

Definition at line 646 of file nifty.py.

Here is the call graph for this function:




### 7.23.3 Variable Documentation

#### 7.23.3.1 float forcebalance::nifty::bohrang = 0.529177249

One bohr equals this many angstroms.

Definition at line 41 of file nifty.py.

#### 7.23.3.2 float forcebalance::nifty::eqcgmx = 2625.5002

Q-Chem to GMX unit conversion for energy.

Definition at line 37 of file nifty.py.

#### 7.23.3.3 float forcebalance::nifty::fqcgmx = 49621.9

Q-Chem to GMX unit conversion for force.

Definition at line 39 of file nifty.py.

#### 7.23.3.4 float forcebalance::nifty::kb = 0.0083144100163

Boltzmann constant.

Definition at line 35 of file nifty.py.

#### 7.23.3.5 tuple forcebalance::nifty::specific_dct = dict(list(itertools.chain(∗[[(j,i[1]) for j in i[0]] for i in specific_lst])))

Definition at line 736 of file nifty.py.

#### 7.23.3.6 list forcebalance::nifty::specific_lst

**Initial value:**

```
1 [(['mdrun','grompp','trjconv','g_energy','g_traj'], "Make sure to install
      GROMACS and add it to your path (or set the gmxpath option)"),
2                (['force.mdin', 'stage.leap'], "This file is needed for setting
      up AMBER force matching targets"),
3                (['conf.pdb', 'mono.pdb'], "This file is needed for setting up
      OpenMM condensed phase property targets"),
4                (['liquid.xyz', 'liquid.key', 'mono.xyz', 'mono.key'], "This
      file is needed for setting up OpenMM condensed phase property targets"),
5                (['dynamic', 'analyze', 'minimize', 'testgrad', 'vibrate', '
     optimize', 'polarize', 'superpose'], "Make sure to install TINKER and add it to
      your path (or set the tinkerpath option)"),
6                (['runcuda.sh', 'npt.py', 'npt_tinker.py'], "This file belongs
      in the ForceBalance source directory, not sure why it is missing"),
7                (['input.xyz'], "This file is needed for TINKER molecular
      property targets"),
8                (['.*key$', '.*xyz$'], "I am guessing this file is probably
      needed by TINKER"),
9                (['.*gro$', '.*top$', '.*itp$', '.*mdp$', '.*ndx$'], "I am
      guessing this file is probably needed by GROMACS")
10                ]
```

Definition at line 724 of file nifty.py.

#### 7.23.3.7 forcebalance::nifty::WORK_QUEUE = None

Definition at line 563 of file nifty.py.

#### 7.23.3.8 tuple forcebalance::nifty::WQIDS = defaultdict(list)

Definition at line 566 of file nifty.py.

#### 7.23.3.9 string forcebalance::nifty::XMLFILE = 'x'

Pickle uses 'flags' to pickle and unpickle different variable types.

Here we use the letter 'x' to signify that the variable type is an XML file.

Definition at line 492 of file nifty.py.

## 7.24 forcebalance::objective Namespace Reference

ForceBalance objective function.

### Classes

- class Objective

  *Objective function.*
- class Penalty

  *Penalty functions for regularizing the force field optimizer.*

### Variables

- dictionary Implemented_Targets

  *The table of implemented Targets.*
- list Letters = ['X','G','H']

  *This is the canonical lettering that corresponds to : objective function, gradient, Hessian.*

#### 7.24.1 Detailed Description

ForceBalance objective function.

#### 7.24.2 Variable Documentation

##### 7.24.2.1 dictionary forcebalance::objective::Implemented_Targets

**Initial value:**

```
1 {
2     'ABINITIO_GMX':AbInitio_GMX,
3     'ABINITIO_TINKER':AbInitio_TINKER,
4     'ABINITIO_OPENMM':AbInitio_OpenMM,
5     'ABINITIO_AMBER':AbInitio_AMBER,
6     'ABINITIO_INTERNAL':AbInitio_Internal,
7     'VIBRATION_TINKER':Vibration_TINKER,
8     'LIQUID_OPENMM':Liquid_OpenMM,
9     'LIQUID_TINKER':Liquid_TINKER,
10     'LIQUID_GMX':Liquid_GMX,
11     'COUNTERPOISE':Counterpoise,
12     'THCDF_PSI4':THCDF_Psi4,
13     'RDVR3_PSI4':RDVR3_Psi4,
14     'INTERACTION_TINKER':Interaction_TINKER,
15     'INTERACTION_OPENMM':Interaction_OpenMM,
16     'BINDINGENERGY_TINKER':BindingEnergy_TINKER,
17     'MOMENTS_TINKER':Moments_TINKER,
18     'MONOMER_QTPIE':Monomer_QTPIE,
19     }
```

The table of implemented Targets.

Definition at line 67 of file objective.py.

##### 7.24.2.2 list forcebalance::objective::Letters = ['X','G','H']

This is the canonical lettering that corresponds to : objective function, gradient, Hessian.

Definition at line 88 of file objective.py.

### 7.25 forcebalance::openmmio Namespace Reference

OpenMM input/output.

#### Classes

- class OpenMM_Reader

  *Class for parsing OpenMM force field files.*
- class Liquid_OpenMM
- class AbInitio_OpenMM

  *Subclass of AbInitio for force and energy matching using OpenMM.*
- class Interaction_OpenMM

  *Subclass of Target for interaction matching using OpenMM.*

### Functions

- def get_dipole

  *Return the current dipole moment in Debye.*
- def ResetVirtualSites

  *Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.*
- def CopyAmoebaBondParameters
- def CopyAmoebaOutOfPlaneBendParameters
- def CopyAmoebaAngleParameters
- def CopyAmoebaInPlaneAngleParameters
- def CopyAmoebaVdwParameters
- def CopyAmoebaMultipoleParameters
- def CopyHarmonicBondParameters
- def CopyHarmonicAngleParameters
- def CopyPeriodicTorsionParameters
- def CopyNonbondedParameters
- def do_nothing
- def CopySystemParameters

  *Copy parameters from one system (i.e.*
- def UpdateSimulationParameters
- def MTSVVVRIntegrator

  *Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.*

### Variables

- dictionary suffix_dict
- string pdict = "XML_Override"

  *pdict is a useless variable if the force field is XML.*

#### 7.25.1 Detailed Description

OpenMM input/output.

**Author**

Lee-Ping Wang

**Date**

04/2012

#### 7.25.2 Function Documentation

##### 7.25.2.1 def forcebalance.openmmio.CopyAmoebaAngleParameters ( *src, dest* )

Definition at line 109 of file openmmio.py.

Here is the call graph for this function:

#### 7.25.2.2 def forcebalance.openmmio.CopyAmoebaBondParameters ( *src, dest* )

Definition at line 95 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.3 def forcebalance.openmmio.CopyAmoebaInPlaneAngleParameters ( *src, dest* )

Definition at line 118 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.4 def forcebalance.openmmio.CopyAmoebaMultipoleParameters ( *src, dest* )

Definition at line 131 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.5 def forcebalance.openmmio.CopyAmoebaOutOfPlaneBendParameters ( *src, dest* )

Definition at line 101 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.6 def forcebalance.openmmio.CopyAmoebaVdwParameters ( *src, dest* )

Definition at line 127 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.7 def forcebalance.openmmio.CopyHarmonicAngleParameters ( *src, dest* )

Definition at line 139 of file openmmio.py.

Here is the call graph for this function:

#### 7.25.2.8 def forcebalance.openmmio.CopyHarmonicBondParameters ( *src, dest* )

Definition at line 135 of file openmmio.py.

Here is the call graph for this function:




#### 7.25.2.9 def forcebalance.openmmio.CopyNonbondedParameters ( *src, dest* )

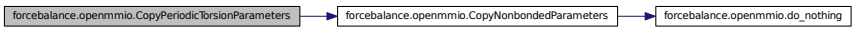
Definition at line 147 of file openmmio.py.

Here is the call graph for this function:


#### 7.25.2.10 def forcebalance.openmmio.CopyPeriodicTorsionParameters ( *src, dest* )

Definition at line 143 of file openmmio.py.

Here is the call graph for this function:



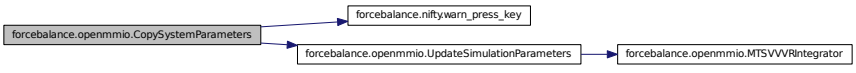

#### 7.25.2.11 def forcebalance.openmmio.CopySystemParameters ( *src, dest* )

Copy parameters from one system (i.e.

that which is created by a new force field) sto another system (i.e. the one stored inside the Target object). DANGER: These need to be implemented manually!!!

Definition at line 161 of file openmmio.py.

Here is the call graph for this function:




#### 7.25.2.12 def forcebalance.openmmio.do_nothing ( *src, dest* )

Definition at line 154 of file openmmio.py.

#### 7.25.2.13 def forcebalance.openmmio.get_dipole ( *simulation, q* = None, *positions* = None )

Return the current dipole moment in Debye.

Note that this quantity is meaningless if the system carries a net charge.

Definition at line 34 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.14 def forcebalance.openmmio.MTSVVVRIntegrator ( *temperature, collision_rate, timestep, system, ninnersteps* = 4 )

Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.

ARGUMENTS

temperature (numpy.unit.Quantity compatible with kelvin) - the temperature collision_rate (numpy.unit.Quantity compatible with 1/picoseconds) - the collision rate timestep (numpy.unit.Quantity compatible with femtoseconds) - the integration timestep system (simtk.openmm.System) - system whose forces will be partitioned ninnersteps (int) - number of inner timesteps (default: 4)

RETURNS

integrator (openmm.CustomIntegrator) - a VVVR integrator

NOTES

This integrator is equivalent to a Langevin integrator in the velocity Verlet discretization with a timestep correction to ensure that the field-free diffusion constant is timestep invariant. The inner velocity Verlet discretization is transformed into a multiple timestep algorithm.

REFERENCES

VVVR Langevin integrator: * http://arxiv.org/abs/1301.3800 * http://arxiv.org/abs/1107.-2967 (to appear in PRX 2013)

TODO

Move initialization of 'sigma' to setting the per-particle variables.

Definition at line 219 of file openmmio.py.

#### 7.25.2.15 def forcebalance.openmmio.ResetVirtualSites ( *positions, system* )

Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.

Definition at line 64 of file openmmio.py.

Here is the call graph for this function:



#### 7.25.2.16 def forcebalance.openmmio.UpdateSimulationParameters ( *src_system, dest_simulation* )

Definition at line 180 of file openmmio.py.

Here is the call graph for this function:


#### 7.25.3 Variable Documentation

##### 7.25.3.1 string forcebalance::openmmio::pdict = "XML_Override"

pdict is a useless variable if the force field is XML.

Definition at line 298 of file openmmio.py.

##### 7.25.3.2 dictionary forcebalance::openmmio::suffix_dict

**Initial value:**

```
1 { "HarmonicBondForce" : {"Bond" : ["class1","class2"]},
2                 "HarmonicAngleForce" : {"Angle" : ["class1","class2","class3"],
      },
3                 "PeriodicTorsionForce" : {"Proper" : ["class1","class2","class3
      ","class4"],},
4                 "NonbondedForce" : {"Atom": ["type"]},
5                 "AmoebaBondForce" : {"Bond" : ["class1","class2"]},
6                 "AmoebaAngleForce" : {"Angle" : ["class1","class2","class3"]},
7                 "AmoebaStretchBendForce" : {"StretchBend" : ["class1","class2",
      "class3"]},
8                 "AmoebaVdwForce" : {"Vdw" : ["class"]},
9                 "AmoebaMultipoleForce" : {"Multipole" : ["type","kz","kx"], "
      Polarize" : ["type"]},
10                 "AmoebaUreyBradleyForce" : {"UreyBradley" : ["class1","class2",
      "class3"]},
11                 "Residues.Residue" : {"VirtualSite" : ["index"]}
12                 }
```

Definition at line 284 of file openmmio.py.

### 7.26 forcebalance::optimizer Namespace Reference

Optimization algorithms.

#### Classes

- class Optimizer

  *Optimizer class.*

#### Functions

- def Counter
- def GoodStep

#### Variables

- int ITERATION_NUMBER = 0
- int GOODSTEP = 0

### 7.26.1 Detailed Description

Optimization algorithms. My current implementation is to have a single optimizer class with several methods contained inside.

#### Author

Lee-Ping Wang

#### Date

12/2011

### 7.26.2 Function Documentation

#### 7.26.2.1 def forcebalance.optimizer.Counter ( )

Definition at line 27 of file optimizer.py.

Here is the call graph for this function:


#### 7.26.2.2 def forcebalance.optimizer.GoodStep ( )

Definition at line 31 of file optimizer.py.

### 7.26.3 Variable Documentation

#### 7.26.3.1 int forcebalance::optimizer::GOODSTEP = 0

Definition at line 25 of file optimizer.py.

#### 7.26.3.2 int forcebalance::optimizer::ITERATION_NUMBER = 0

Definition at line 23 of file optimizer.py.

## 7.27 forcebalance::parser Namespace Reference

Input file parser for ForceBalance jobs.

#### Functions

- def read_mvals
- def read_pvals
- def read_priors
- def read_internals
- def printsection

  *Print out a section of the input file in a parser-compliant and readable format.*
- def parse_inputs

  *Parse through the input file and read all user-supplied options.*

#### Variables

- dictionary gen_opts_types

  *Default general options.*
- dictionary tgt_opts_types

  *Default fitting target options.*
- dictionary gen_opts_defaults = {}

  *Default general options - basically a collapsed veresion of gen_opts_types.*
- dictionary subdict = {}
- dictionary tgt_opts_defaults = {}

  *Default target options - basically a collapsed version of tgt_opts_types.*
- dictionary bkwd = {"simtype" : "type"}

  *Option maps for maintaining backward compatibility.*
- list mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

  *Listing of sections in the input file.*
- dictionary ParsTab

  *ParsTab that refers to subsection parsers.*

### 7.27.1 Detailed Description

Input file parser for ForceBalance jobs. Additionally, the location for all default options.

Although I will do my best to write good documentation, for many programs the input parser becomes the most up-to-date source for documentation. So this is a great place to write lots of comments for those who implement new functionality.

There are two types of sections for options - GENERAL and TARGET. Since there can be many fitting targets within a single job (i.e. we may wish to fit water trimers and hexamers, which constitutes two fitting targets) the input is organized into sections, like so:

$options

gen_option_1 Big

gen_option_2 Mao

$target

tgt_option_1 Sniffy

tgt_option_2 Schmao

$target

tgt_option_1 Nifty

tgt_option_2 Jiffy

$end

In this case, two sets of target options are generated in addition to the general option.

(Note: "Target" used to be called "Simulation". Backwards compatibility is maintained.)

Each option is meant to be parsed as a certain variable type.

- String option values are read in directly; note that only the first two words in the line are processed
- Some strings are capitalized when they are read in; this is mainly for function tables like OptTab and TgtTab
- List option types will pick up all of the words on the line and use them as values, plus if the option occurs more than once it will aggregate all of the values.
- Integer and float option types are read in a pretty straightforward way
- Boolean option types are always set to true, unless the second word is '0', 'no', or 'false' (not case sensitive)
- Section option types are meant to treat more elaborate inputs, such as the user pasting in output parameters from a previous job as input, or a specification of internal coordinate system. I imagine that for every section type I would have to write my own parser. Maybe a ParsTab of parsing functions would work. :)

To add a new option, simply add it to the dictionaries below and give it a default value if desired. If you add an entirely new type, make sure to implement the interpretation of that type in the parse_inputs function.

**Author**

Lee-Ping Wang

**Date**

11/2012

### 7.27.2 Function Documentation

#### 7.27.2.1 def forcebalance.parser.parse_inputs ( *input_file* = `None` )

Parse through the input file and read all user-supplied options.

This is usually the first thing that happens when an executable script is called. Our parser first loads the default options, and then updates these options as it encounters keywords.

Each keyword corresponds to a variable type; each variable type (e.g. string, integer, float, boolean) is treated differently. For more elaborate inputs, there is a 'section' variable type.

There is only one set of general options, but multiple sets of target options. Each target has its own section delimited by the *$target* keyword, and we build a list of target options.

**Parameters**

| | | |
|---|---|---|
| `in` | *input_file* | The name of the input file. |

**Returns**

options General options.
tgt_opts List of fitting target options.

**Todo** Implement internal coordinates.

Implement sampling correction.

Implement charge groups.

Definition at line 372 of file parser.py.

Here is the call graph for this function:

#### 7.27.2.2 def forcebalance.parser.printsection ( *heading, optdict, typedict* )

Print out a section of the input file in a parser-compliant and readable format.

At the time of writing of this function, it's mainly intended to be called by MakeInputFile.py. The heading is printed first (it is something like $options or $target). Then it loops through the variable types (strings, allcaps, etc...) and the keys in each variable type. The one-line description of each key is printed out as a comment, and then the key itself is printed out along with the value provided in optdict. If optdict is None, then the default value is printed out instead.

**Parameters**

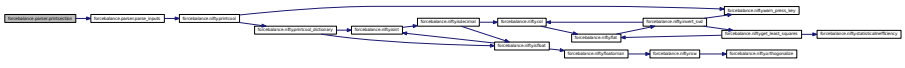
| | | |
|---|---|---|
| in | *heading* | Heading, either $options or $target |
| in | *optdict* | Options dictionary or None. |
| in | *typedict* | Option type dictionary, either gen_opts_types or tgt_opts_types specified in this file. |

**Returns**

Answer List of strings for the section that we are printing out.

Definition at line 275 of file parser.py.

Here is the call graph for this function:



#### 7.27.2.3 def forcebalance.parser.read_internals ( *fobj* )

Definition at line 249 of file parser.py.

#### 7.27.2.4 def forcebalance.parser.read_mvals ( *fobj* )

Definition at line 224 of file parser.py.

Here is the call graph for this function:


#### 7.27.2.5 def forcebalance.parser.read_priors ( *fobj* )

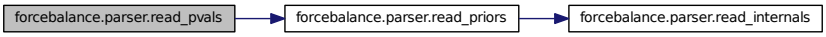
Definition at line 240 of file parser.py.

Here is the call graph for this function:


#### 7.27.2.6 def forcebalance.parser.read_pvals ( *fobj* )

Definition at line 232 of file parser.py.

Here is the call graph for this function:




### 7.27.3 Variable Documentation

#### 7.27.3.1 dictionary forcebalance::parser::bkwd = {"simtype" : "type"}

Option maps for maintaining backward compatibility.

Definition at line 219 of file parser.py.

#### 7.27.3.2 dictionary forcebalance::parser::gen_opts_defaults = {}

Default general options - basically a collapsed veresion of gen_opts_types.

Definition at line 203 of file parser.py.

#### 7.27.3.3 dictionary forcebalance::parser::gen_opts_types

Default general options.

Note that the documentation is included in part of the key; this will aid in automatic doc-extraction. :) In the 5-tuple we have: Default value, priority (larger number means printed first), short docstring, description of scope, list of filter strings for pulling out pertinent targets (MakeInputFile.py)

Definition at line 62 of file parser.py.

#### 7.27.3.4 list forcebalance::parser::mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

Listing of sections in the input file.

Definition at line 222 of file parser.py.

#### 7.27.3.5 dictionary forcebalance::parser::ParsTab

**Initial value:**

```
1 {"read_mvals" : read_mvals,
2             "read_pvals" : read_pvals,
3             "priors"     : read_priors,
4             "internal"   : read_internals
5             }
```

ParsTab that refers to subsection parsers.

Definition at line 253 of file parser.py.

#### 7.27.3.6 dictionary forcebalance::parser::subdict = {}

Definition at line 205 of file parser.py.

#### 7.27.3.7 dictionary forcebalance::parser::tgt_opts_defaults = {}

Default target options - basically a collapsed version of tgt_opts_types.

Definition at line 211 of file parser.py.

#### 7.27.3.8 dictionary forcebalance::parser::tgt_opts_types

Default fitting target options.

Definition at line 123 of file parser.py.

## 7.28 forcebalance::psi4io Namespace Reference

PSI4 force field input/output.

### Classes

- class GBS_Reader
  *Interaction type -> Parameter Dictionary.*
- class THCDF_Psi4
- class Grid_Reader
  *Finite state machine for parsing DVR grid files.*
- class RDVR3_Psi4
  *Subclass of Target for R-DVR3 grid fitting.*

### 7.28.1 Detailed Description

PSI4 force field input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

**Author**

Lee-Ping Wang

**Date**

01/2012

## 7.29 forcebalance::PT Namespace Reference

**Variables**

- dictionary PeriodicTable
- list Elements

### 7.29.1 Variable Documentation

#### 7.29.1.1 list forcebalance::PT::Elements

**Initial value:**

```
1 ["None",'H','He',
2             'Li','Be','B','C','N','O','F','Ne',
3             'Na','Mg','Al','Si','P','S','Cl','Ar',
4             'K','Ca','Sc','Ti','V','Cr','Mn','Fe','Co','Ni','Cu','Zn','Ga','Ge'
      ,'As','Se','Br','Kr',
5             'Rb','Sr','Y','Zr','Nb','Mo','Tc','Ru','Rh','Pd','Ag','Cd','In','Sn
      ','Sb','Te','I','Xe',
6             'Cs','Ba','La','Ce','Pr','Nd','Pm','Sm','Eu','Gd','Tb','Dy','Ho','
      Er','Tm','Yb',
7             'Lu','Hf','Ta','W','Re','Os','Ir','Pt','Au','Hg','Tl','Pb','Bi','Po
      ','At','Rn',
8             'Fr','Ra','Ac','Th','Pa','U','Np','Pu','Am','Cm','Bk','Cf','Es','Fm
      ','Md','No','Lr','Rf','Db','Sg','Bh','Hs','Mt']
```

Definition at line 18 of file PT.py.

#### 7.29.1.2 dictionary forcebalance::PT::PeriodicTable

**Initial value:**

```
1 {'H' : 1.0079, 'He' : 4.0026,
2               'Li' : 6.941, 'Be' : 9.0122, 'B' : 10.811, 'C' : 12.0107, 'N'
      : 14.0067, 'O' : 15.9994, 'F' : 18.9984, 'Ne' : 20.1797,
3               'Na' : 22.9897, 'Mg' : 24.305, 'Al' : 26.9815, 'Si' : 28.0855,
       'P' : 30.9738, 'S' : 32.065, 'Cl' : 35.453, 'Ar' : 39.948,
4               'K' : 39.0983, 'Ca' : 40.078, 'Sc' : 44.9559, 'Ti' : 47.867, '
      V' : 50.9415, 'Cr' : 51.9961, 'Mn' : 54.938, 'Fe' : 55.845, 'Co' : 58.9332,
5               'Ni' : 58.6934, 'Cu' : 63.546, 'Zn' : 65.39, 'Ga' : 69.723, '
      Ge' : 72.64, 'As' : 74.9216, 'Se' : 78.96, 'Br' : 79.904, 'Kr' : 83.8,
6               'Rb' : 85.4678, 'Sr' : 87.62, 'Y' : 88.9059, 'Zr' : 91.224, '
      Nb' : 92.9064, 'Mo' : 95.94, 'Tc' : 98, 'Ru' : 101.07, 'Rh' : 102.9055,
7               'Pd' : 106.42, 'Ag' : 107.8682, 'Cd' : 112.411, 'In' : 114.818
      , 'Sn' : 118.71, 'Sb' : 121.76, 'Te' : 127.6, 'I' : 126.9045, 'Xe' : 131.293,
8               'Cs' : 132.9055, 'Ba' : 137.327, 'La' : 138.9055, 'Ce' : 140.1
      16, 'Pr' : 140.9077, 'Nd' : 144.24, 'Pm' : 145, 'Sm' : 150.36,
9               'Eu' : 151.964, 'Gd' : 157.25, 'Tb' : 158.9253, 'Dy' : 162.5,
      'Ho' : 164.9303, 'Er' : 167.259, 'Tm' : 168.9342, 'Yb' : 173.04,
10               'Lu' : 174.967, 'Hf' : 178.49, 'Ta' : 180.9479, 'W' : 183.84,
      'Re' : 186.207, 'Os' : 190.23, 'Ir' : 192.217, 'Pt' : 195.078,
```

```
11                  'Au' : 196.9665, 'Hg' : 200.59, 'Tl' : 204.3833, 'Pb' : 207.2,
       'Bi' : 208.9804, 'Po' : 209, 'At' : 210, 'Rn' : 222,
12                  'Fr' : 223, 'Ra' : 226, 'Ac' : 227, 'Th' : 232.0381, 'Pa' : 23
      1.0359, 'U' : 238.0289, 'Np' : 237, 'Pu' : 244,
13                  'Am' : 243, 'Cm' : 247, 'Bk' : 247, 'Cf' : 251, 'Es' : 252, '
      Fm' : 257, 'Md' : 258, 'No' : 259,
14                  'Lr' : 262, 'Rf' : 261, 'Db' : 262, 'Sg' : 266, 'Bh' : 264, '
      Hs' : 277, 'Mt' : 268}
```

Definition at line 3 of file PT.py.

## 7.30 forcebalance::qchemio Namespace Reference

Q-Chem input file parser.

### Classes

- class QCIn_Reader

  *Finite state machine for parsing Q-Chem input files.*

### Functions

- def QChem_Dielectric_Energy

### Variables

- list ndtypes = [None]

  *Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.*
- dictionary pdict

  *Section -> Interaction type dictionary.*

#### 7.30.1 Detailed Description

Q-Chem input file parser.

#### 7.30.2 Function Documentation

##### 7.30.2.1 def forcebalance.qchemio.QChem_Dielectric_Energy ( *fnm, wq* )

Definition at line 71 of file qchemio.py.

Here is the call graph for this function:

#### 7.30.3 Variable Documentation

##### 7.30.3.1 list forcebalance::qchemio::ndtypes = [None]

Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.

Definition at line 13 of file qchemio.py.

##### 7.30.3.2 dictionary forcebalance::qchemio::pdict

**Initial value:**

```
1 {'BASS':{0:'A', 1:'C'},
2          'BASSP' :{0:'A', 1:'B', 2:'C'}
3          }
```

Section -> Interaction type dictionary.

fdict = { 'basis' : bastypes } Interaction type -> Parameter Dictionary.

Definition at line 20 of file qchemio.py.

### 7.31 forcebalance::target Namespace Reference

#### Classes

- class Target

  *Base class for all fitting targets.*

### 7.32 forcebalance::tinkerio Namespace Reference

TINKER input/output.

#### Classes

- class Tinker_Reader

  *Finite state machine for parsing TINKER force field files.*
- class Liquid_TINKER
- class AbInitio_TINKER

  *Subclass of Target for force and energy matching using TINKER.*
- class Vibration_TINKER

  *Subclass of Target for vibrational frequency matching using TINKER.*
- class Moments_TINKER

  *Subclass of Target for multipole moment matching using TINKER.*
- class BindingEnergy_TINKER

  *Subclass of BindingEnergy for binding energy matching using TINKER.*
- class Interaction_TINKER

  *Subclass of Target for interaction matching using TINKER.*

#### Functions

- def write_key_with_prm

  *Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.*
- def modify_key

  *Performs in-place modification of a TINKER .key file.*

#### Variables

- dictionary pdict

### 7.32.1 Detailed Description

TINKER input/output. This serves as a good template for writing future force matching I/O modules for other programs because it's so simple.

#### Author

Lee-Ping Wang

#### Date

01/2012

### 7.32.2 Function Documentation

#### 7.32.2.1 def forcebalance.tinkerio.modify_key ( *src, in_dict* )

Performs in-place modification of a TINKER .key file.

The input dictionary contains key:value pairs such as "polarization direct". If the key exists in the TINKER file, then that line is modified such that it contains the value in the dictionary. Note that this "key" is not to be confused with the .key extension in the TINKER file that we're modifying.

Sometimes keys like 'archive' do not have a value, in which case the dictionary should contain a None value or a blank space.

If the key doesn't exist in the TINKER file, then the key:value pair will be printed at the end.

**Parameters**

| | | |
|---|---|---|
| in | *src* | Name of the TINKER file to be modified. |
| in | *in_dict* | Dictionary containing key-value pairs used to modify the TINKER file. |

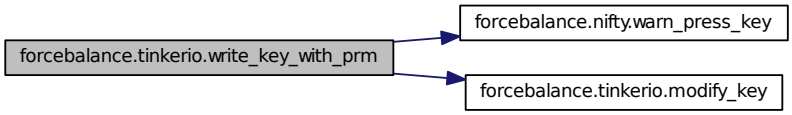
Definition at line 199 of file tinkerio.py.

#### 7.32.2.2 def forcebalance.tinkerio.write_key_with_prm ( *src, dest, prmfnm* = None, *ffobj* = None )

Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.

Definition at line 143 of file tinkerio.py.

Here is the call graph for this function:




#### 7.32.3 Variable Documentation

##### 7.32.3.1 dictionary forcebalance::tinkerio::pdict

**Initial value:**

```
1 {'VDW'          : {'Atom':[1], 2:'S',3:'T',4:'D'}, # Van der Waals distance,
       well depth, distance from bonded neighbor?
2          'BOND'         : {'Atom':[1,2], 3:'K',4:'B'},     # Bond force
       constant and equilibrium distance (Angstrom)
3          'ANGLE'        : {'Atom':[1,2,3], 4:'K',5:'B'},   # Angle force
       constant and equilibrium angle
4          'UREYBRAD'     : {'Atom':[1,2,3], 4:'K',5:'B'},   # Urey-Bradley force
       constant and equilibrium distance (Angstrom)
5          'MCHARGE'       : {'Atom':[1,2,3], 4:''},          # Atomic charge
6          'DIPOLE'       : {0:'X',1:'Y',2:'Z'},             # Dipole moment in
       local frame
7          'QUADX'        : {0:'X'},                         # Quadrupole moment,
       X component
8          'QUADY'        : {0:'X',1:'Y'},                   # Quadrupole moment,
       Y component
9          'QUADZ'        : {0:'X',1:'Y',2:'Z'},             # Quadrupole moment,
       Y component
10          'POLARIZE'     : {'Atom':[1], 2:'A',3:'T'},       # Atomic dipole
       polarizability
11          'BOND-CUBIC'   : {'Atom':[], 0:''},    # Below are global parameters.
12          'BOND-QUARTIC' : {'Atom':[], 0:''},
13          'ANGLE-CUBIC'  : {'Atom':[], 0:''},
14          'ANGLE-QUARTIC': {'Atom':[], 0:''},
15          'ANGLE-PENTIC' : {'Atom':[], 0:''},
16          'ANGLE-SEXTIC' : {'Atom':[], 0:''},
17          'DIELECTRIC'   : {'Atom':[], 0:''},
18          'POLAR-SOR'    : {'Atom':[], 0:''}
19                                                 # Ignored for now: stretch/bend
       coupling, out-of-plane bending,
20                                                 # torsional parameters,
       pi-torsion, torsion-torsion
21          }
```

Definition at line 31 of file tinkerio.py.

### 7.33 forcebalance::vibration Namespace Reference

Vibrational mode fitting module.

#### Classes

- class Vibration

  *Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).*

### 7.33.1 Detailed Description

Vibrational mode fitting module.

#### Author

Lee-Ping Wang

#### Date

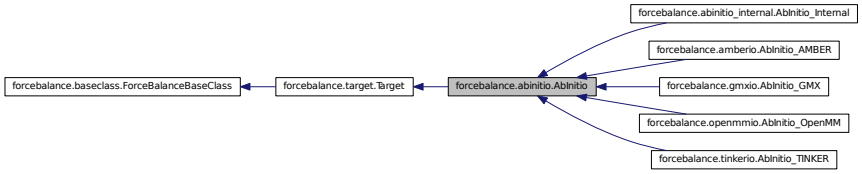
08/2012

# 8 Class Documentation

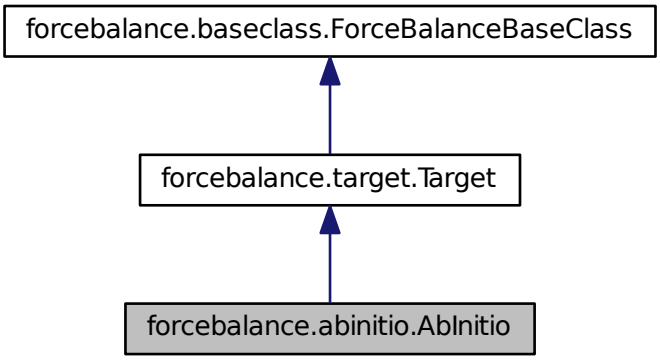
## 8.1 forcebalance.abinitio.AbInitio Class Reference

Subclass of Target for fitting force fields to ab initio data.

Inheritance diagram for forcebalance.abinitio.AbInitio:



Collaboration diagram for forcebalance.abinitio.AbInitio:




**Public Member Functions**

- def __init__

  *Initialization; define a few core concepts.*
- def read_topology
- def build_invdist
- def compute_netforce_torque
- def read_reference_data

  *Read the reference ab initio data from a file such as qdata.txt.*
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def indicate
- def energy_force_transformer_all
- def energy_force_transformer
- def get_energy_force_

  *LPW 06-30-2013.*
- def get_resp_

  *Electrostatic potential fitting.*
- def get

  *Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.*

**Public Attributes**

- whamboltz_wts

  *Initialize the base class.*
- qmboltz_wts

  *QM Boltzmann weights.*

- eqm

  *Reference (QM) energies.*
- emd0

  *Energies of the sampling simulation.*
- fqm

  *Reference (QM) forces.*
- espxyz

  *ESP grid points.*
- espval

  *ESP values.*
- qfnm

  *The qdata.txt file that contains the QM energies and forces.*
- qmatoms

  *The number of atoms in the QM calculation (Irrelevant if not fitting forces)*
- e_err

  *Qualitative Indicator: average energy error (in kJ/mol)*
- e_err_pct
- f_err

  *Qualitative Indicator: average force error (fractional)*
- f_err_pct
- esp_err

  *Qualitative Indicator: "relative RMS" for electrostatic potential.*
- nf_err
- nf_err_pct
- tq_err_pct
- use_nft

  *Whether to compute net forces and torques, or not.*
- ns

  *Read in the trajectory file.*
- traj
- nparticles

  *The number of (atoms + drude particles + virtual sites)*
- AtomLists

  *This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.*
- new_vsites

  *Read in the topology.*
- save_vmvals

  *Save the mvals from the last time we updated the vsites.*
- topology_flag
- force_map
- nnf
- ntq
- force
- w_force
- nesp
- fitatoms
- whamboltz

- nftqm
- fref
- w_energy
- w_netforce
- w_torque
- e_ref
- f_ref
- nf_ref
- tq_ref
- tq_err
- w_resp
- invdists
- respterm
- objective

#### 8.1.1 Detailed Description

Subclass of Target for fitting force fields to ab initio data.

Currently Gromacs-X2, Gromacs, Tinker, OpenMM and AMBER are supported.

We introduce the following concepts:

- The number of snapshots
- The reference energies and forces (eqm, fqm) and the file they belong in (qdata.txt)
- The sampling simulation energies (emd0)
- The WHAM Boltzmann weights (these are computed externally and passed in)
- The QM Boltzmann weights (computed internally using the difference between eqm and emd0)

There are also these little details:

- Switches for whether to turn on certain Boltzmann weights (they stack)
- Temperature for the QM Boltzmann weights
- Whether to fit a subset of atoms

This subclass contains the 'get' method for building the objective function from any simulation software (a driver to run the program and read output is still required). The 'get' method can be overridden by subclasses like AbInitio_GMX.

Definition at line 44 of file abinitio.py.

#### 8.1.2 Constructor & Destructor Documentation

##### 8.1.2.1 def forcebalance.abinitio.AbInitio.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization; define a few core concepts.

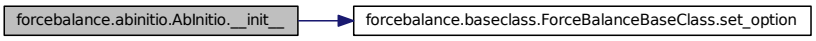
**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.gmxio.AbInitio_GMX, forcebalance.openmmio.AbInitio_OpenMM, forcebalance.tinkerio.AbInitio_TINKER, forcebalance.amberio.AbInitio_AMBER, and forcebalance.abinitio_internal.AbInitio_Internal.

Definition at line 54 of file abinitio.py.

Here is the call graph for this function:

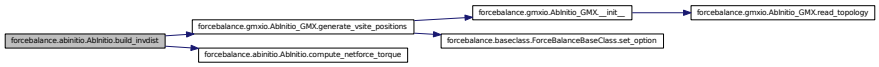



### 8.1.3 Member Function Documentation

#### 8.1.3.1 def forcebalance.abinitio.AbInitio.build_invdist ( *self, mvals* )

Definition at line 165 of file abinitio.py.

Here is the call graph for this function:



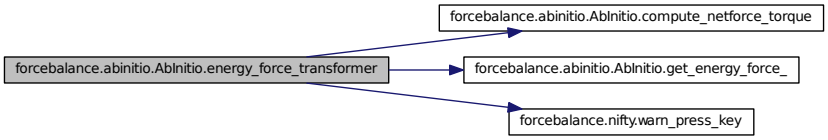

#### 8.1.3.2 def forcebalance.abinitio.AbInitio.compute_netforce_torque ( *self, xyz, force, QM* = `False` )

Definition at line 201 of file abinitio.py.

#### 8.1.3.3 def forcebalance.abinitio.AbInitio.energy_force_transformer ( *self, i* )

Definition at line 455 of file abinitio.py.

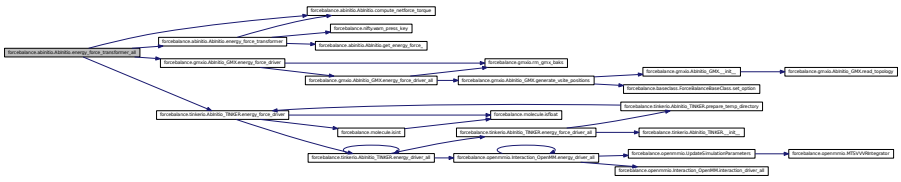
Here is the call graph for this function:




#### 8.1.3.4 def forcebalance.abinitio.AbInitio.energy_force_transformer_all ( *self* )

Definition at line 439 of file abinitio.py.

Here is the call graph for this function:



#### 8.1.3.5 def forcebalance.abinitio.AbInitio.get ( *self, mvals, AGrad =* `False`*, AHess =* `False` )

Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.

See abinitio for an example.

Reimplemented from forcebalance.target.Target.

Definition at line 1095 of file abinitio.py.

#### 8.1.3.6 def forcebalance.abinitio.AbInitio.get_energy_force_ ( *self, mvals, AGrad =* `False`*, AHess =* `False` )

LPW 06-30-2013.

This subroutine builds the objective function (and optionally its derivatives) from a general simulation software. This is in contrast to using GROMACS-X2, which computes the objective function and prints it out; then 'get' only needs to call GROMACS and read it in.

This subroutine interfaces with simulation software 'drivers'. The driver is only expected to give the energy and forces.

Now this subroutine may sound trivial since the objective function is simply a least-squares quantity (M-Q)^2 - but there are a number of nontrivial considerations. I will list them here.

0) Polytensor formulation: Because there may exist covariance between different components of the force (or covariance between the energy and the force), we build the objective function by taking outer products of vectors that have the form [E F_1x F_1y F_1z F_2x F_2y ... ], and then we trace it with the inverse of the covariance matrix to get the objective function.

This version implements both the polytensor formulation and the standard formulation.

1) Boltzmann weights and normalization: Each snapshot has its own Boltzmann weight, which may or may not be normalized. This subroutine does the normalization automatically.

2) Subtracting out the mean energy gap: The zero-point energy difference between reference data and simulation is meaningless. This subroutine subtracts it out.

3) Hybrid ensembles: This program builds a combined objective function from both MM and QM ensembles, which is rigorously better than using a single ensemble.

Note that this subroutine does not do EVERYTHING that GROMACS-X2 can do, which includes:

1) Internal coordinate systems 2) 'Sampling correction' (deprecated, since it doesn't seem to work) 3) Analytic derivatives

In the previous code (ForTune) this subroutine used analytic first derivatives of the energy and force to build the derivatives of the objective function. Here I will take a simplified approach, because building the derivatives are cumbersome. For now we will return the objective function ONLY. A two-point central difference should give us the first and diagonal second derivative anyhow.

**Todo** Parallelization over snapshots is not implemented yet

**Parameters**

| | | |
|---|---|---|
| in | *mvals* | Mathematical parameter values |
| in | *AGrad* | Switch to turn on analytic gradient |
| in | *AHess* | Switch to turn on analytic Hessian |

**Returns**

Answer Contribution to the objective function

Definition at line 529 of file abinitio.py.

#### 8.1.3.7 def forcebalance.abinitio.AbInitio.get_resp_ ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Electrostatic potential fitting.

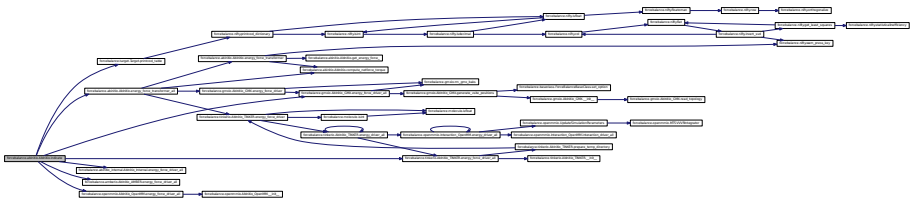
Implements the RESP objective function. (In Python so obviously not optimized.) This function takes the mathematical parameter values and returns the charges on the ATOMS (fancy mapping going on)

Definition at line 996 of file abinitio.py.

#### 8.1.3.8 def forcebalance.abinitio.AbInitio.indicate ( *self* )

Definition at line 419 of file abinitio.py.

Here is the call graph for this function:



#### 8.1.3.9 def forcebalance.abinitio.AbInitio.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented in forcebalance.gmxio.AbInitio_GMX, forcebalance.openmmio.AbInitio_OpenMM, forcebalance.tinkerio.AbInitio_TINKER, and forcebalance.amberio.AbInitio_AMBER.

Definition at line 416 of file abinitio.py.

#### 8.1.3.10 def forcebalance.abinitio.AbInitio.read_reference_data ( *self* )

Read the reference ab initio data from a file such as qdata.txt.

**Todo** Add an option for picking any slice out of qdata.txt, helpful for cross-validation

**Todo** Closer integration of reference data with program - leave behind the qdata.txt format? (For now, I like the readability of qdata.txt)

After reading in the information from qdata.txt, it is converted into the GROMACS energy units (kind of an arbitrary choice); forces (kind of a misnomer in qdata.txt) are multipled by -1 to convert gradients to forces.

We also subtract out the mean energies of all energy arrays because energy/force matching does not account for zero-point energy differences between MM and QM (i.e. energy of electrons in core orbitals).

The configurations in force/energy matching typically come from a the thermodynamic ensemble of the MM force field at some temperature (by running MD, for example), and for many reasons it is helpful to introduce non-Boltzmann weights in front of these configurations. There are two options: WHAM Boltzmann weights (for combining the weights of several simulations together) and QM Boltzmann weights (for converting MM weights into QM weights). Note that the two sets of weights 'stack'; i.e. they can be used at the same time.

A 'hybrid' ensemble is possible where we use 50% MM and 50% QM weights. Please read more in LPW and Troy Van Voorhis, JCP Vol. 133, Pg. 231101 (2010), doi:10.1063/1.3519043.

**Todo** The WHAM Boltzmann weights are generated by external scripts (wanalyze.py and make-wham-data.sh) and passed in; perhaps these scripts can be added to the ForceBalance distribution or integrated more tightly.

Finally, note that using non-Boltzmann weights degrades the statistical information content of the snapshots. This problem will generally become worse if the ensemble to which we're reweighting is dramatically different from the one we're sampling from. We end up with a set of Boltzmann weights like [1e-9, 1e-9, 1.0, 1e-9, 1e-9 ... ] and this is essentially just one snapshot. I believe Troy is working on something to cure this problem.

Here, we have a measure for the information content of our snapshots, which comes easily from the definition of information entropy:

S = -1∗Sum_i(P_i∗log(P_i)) InfoContent = exp(-S)

With uniform weights, InfoContent is equal to the number of snapshots; with horrible weights, InfoContent is closer to one.

Definition at line 314 of file abinitio.py.

#### 8.1.3.11 def forcebalance.abinitio.AbInitio.read_topology ( *self* )

Reimplemented in forcebalance.openmmio.AbInitio_OpenMM, and forcebalance.gmxio.AbInitio_GMX.

Definition at line 161 of file abinitio.py.

### 8.1.4 Member Data Documentation

#### 8.1.4.1 forcebalance::abinitio.AbInitio::AtomLists

This is a default-dict containing a number of atom-wise lists, such as the residue number of each atom, the mass of each atom, and so on.

Definition at line 89 of file abinitio.py.

#### 8.1.4.2 forcebalance::abinitio.AbInitio::e_err

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 82 of file abinitio.py.

#### 8.1.4.3 forcebalance::abinitio.AbInitio::e_err_pct

Definition at line 82 of file abinitio.py.

#### 8.1.4.4 forcebalance::abinitio.AbInitio::e_ref

Definition at line 529 of file abinitio.py.

#### 8.1.4.5 forcebalance::abinitio.AbInitio::emd0

Energies of the sampling simulation.

Definition at line 76 of file abinitio.py.

#### 8.1.4.6 forcebalance::abinitio.AbInitio::eqm

Reference (QM) energies.

Definition at line 75 of file abinitio.py.

#### 8.1.4.7 forcebalance::abinitio.AbInitio::esp_err

Qualitative Indicator: "relative RMS" for electrostatic potential.

Definition at line 84 of file abinitio.py.

#### 8.1.4.8 forcebalance::abinitio.AbInitio::espval

ESP values.

Definition at line 79 of file abinitio.py.

#### 8.1.4.9 forcebalance::abinitio.AbInitio::espxyz

ESP grid points.

Definition at line 78 of file abinitio.py.

#### 8.1.4.10 forcebalance::abinitio.AbInitio::f_err

Qualitative Indicator: average force error (fractional)

Definition at line 83 of file abinitio.py.

#### 8.1.4.11 forcebalance::abinitio.AbInitio::f_err_pct

Definition at line 83 of file abinitio.py.

#### 8.1.4.12 forcebalance::abinitio.AbInitio::f_ref

Definition at line 529 of file abinitio.py.

#### 8.1.4.13 forcebalance::abinitio.AbInitio::fitatoms

Definition at line 314 of file abinitio.py.

#### 8.1.4.14 forcebalance::abinitio.AbInitio::force

Definition at line 314 of file abinitio.py.

#### 8.1.4.15 forcebalance::abinitio.AbInitio::force_map

Definition at line 201 of file abinitio.py.

#### 8.1.4.16 forcebalance::abinitio.AbInitio::fqm

Reference (QM) forces.

Definition at line 77 of file abinitio.py.

#### 8.1.4.17 forcebalance::abinitio.AbInitio::fref

Definition at line 314 of file abinitio.py.

#### 8.1.4.18 forcebalance::abinitio.AbInitio::invdists

Definition at line 996 of file abinitio.py.

#### 8.1.4.19 forcebalance::abinitio.AbInitio::nesp

Definition at line 314 of file abinitio.py.

#### 8.1.4.20 forcebalance::abinitio.AbInitio::new_vsites

Read in the topology.

Read in the reference data Prepare the temporary directory The below two options are related to whether we want to rebuild virtual site positions. Rebuild the distance matrix if virtual site positions have changed

Definition at line 94 of file abinitio.py.

#### 8.1.4.21 forcebalance::abinitio.AbInitio::nf_err

Definition at line 84 of file abinitio.py.

#### 8.1.4.22 forcebalance::abinitio.AbInitio::nf_err_pct

Definition at line 84 of file abinitio.py.

#### 8.1.4.23 forcebalance::abinitio.AbInitio::nf_ref

Definition at line 529 of file abinitio.py.

#### 8.1.4.24 forcebalance::abinitio.AbInitio::nftqm

Definition at line 314 of file abinitio.py.

#### 8.1.4.25 forcebalance::abinitio.AbInitio::nnf

Definition at line 201 of file abinitio.py.

#### 8.1.4.26 forcebalance::abinitio.AbInitio::nparticles

The number of (atoms + drude particles + virtual sites)

Definition at line 87 of file abinitio.py.

#### 8.1.4.27 forcebalance::abinitio.AbInitio::ns

Read in the trajectory file.

Definition at line 86 of file abinitio.py.

#### 8.1.4.28 forcebalance::abinitio.AbInitio::ntq

Definition at line 201 of file abinitio.py.

#### 8.1.4.29 forcebalance::abinitio.AbInitio::objective

Definition at line 1095 of file abinitio.py.

#### 8.1.4.30 forcebalance::abinitio.AbInitio::qfnm

The qdata.txt file that contains the QM energies and forces.

Definition at line 80 of file abinitio.py.

#### 8.1.4.31 forcebalance::abinitio.AbInitio::qmatoms

The number of atoms in the QM calculation (Irrelevant if not fitting forces)

Definition at line 81 of file abinitio.py.

#### 8.1.4.32 forcebalance::abinitio.AbInitio::qmboltz_wts

QM Boltzmann weights.

Definition at line 74 of file abinitio.py.

#### 8.1.4.33 forcebalance::abinitio.AbInitio::respterm

Definition at line 996 of file abinitio.py.

#### 8.1.4.34 forcebalance::abinitio.AbInitio::save_vmvals

Save the mvals from the last time we updated the vsites.

Definition at line 95 of file abinitio.py.

#### 8.1.4.35 forcebalance::abinitio.AbInitio::topology_flag

Reimplemented in forcebalance.openmmio.AbInitio_OpenMM, and forcebalance.gmxio.AbInitio_GMX.

Definition at line 161 of file abinitio.py.

#### 8.1.4.36 forcebalance::abinitio.AbInitio::tq_err

Definition at line 529 of file abinitio.py.

#### 8.1.4.37 forcebalance::abinitio.AbInitio::tq_err_pct

Definition at line 84 of file abinitio.py.

#### 8.1.4.38 forcebalance::abinitio.AbInitio::tq_ref

Definition at line 529 of file abinitio.py.

#### 8.1.4.39 forcebalance::abinitio.AbInitio::traj

Definition at line 86 of file abinitio.py.

#### 8.1.4.40 forcebalance::abinitio.AbInitio::use_nft

Whether to compute net forces and torques, or not.

Definition at line 85 of file abinitio.py.

#### 8.1.4.41 forcebalance::abinitio.AbInitio::w_energy

Definition at line 529 of file abinitio.py.

#### 8.1.4.42 forcebalance::abinitio.AbInitio::w_force

Definition at line 314 of file abinitio.py.

#### 8.1.4.43 forcebalance::abinitio.AbInitio::w_netforce

Definition at line 529 of file abinitio.py.

#### 8.1.4.44 forcebalance::abinitio.AbInitio::w_resp

Definition at line 996 of file abinitio.py.

#### 8.1.4.45 forcebalance::abinitio.AbInitio::w_torque

Definition at line 529 of file abinitio.py.

#### 8.1.4.46 forcebalance::abinitio.AbInitio::whamboltz

Definition at line 314 of file abinitio.py.

#### 8.1.4.47 forcebalance::abinitio.AbInitio::whamboltz_wts

Initialize the base class.

Number of snapshots Whether to use WHAM Boltzmann weights Whether to use the Sampling Correction Whether to match Absolute Energies (make sure you know what you're doing) Whether to use the Covariance Matrix Whether to use QM Boltzmann weights The temperature for QM Boltzmann weights Number of atoms that we are fitting Whether to fit Energies. Whether to fit Forces. Whether to fit Electrostatic Potential. Weights for the three components. Option for how much data to write to disk. Whether to do energy and force calculations for the whole trajectory, or to do one calculation per snapshot. OpenMM-only option - whether to run the energies and forces internally. Whether we have virtual sites (set at the global option level) WHAM Boltzmann weights

Definition at line 73 of file abinitio.py.

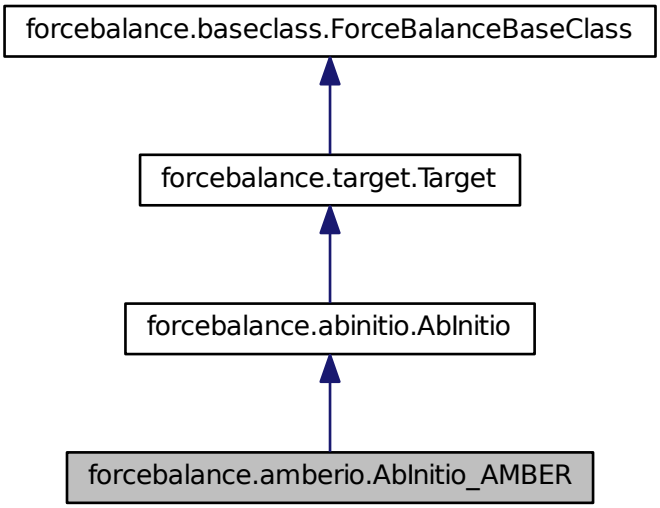
The documentation for this class was generated from the following file:

- abinitio.py

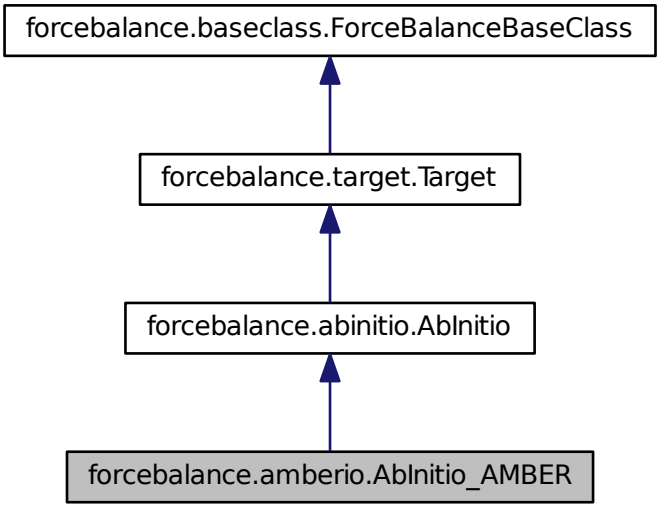
## 8.2 forcebalance.amberio.AbInitio_AMBER Class Reference

Subclass of Target for force and energy matching using AMBER.

Inheritance diagram for forcebalance.amberio.AbInitio_AMBER:




Collaboration diagram for forcebalance.amberio.AbInitio_AMBER:



#### Public Member Functions

- def __init__
  *Initialization; define a few core concepts.*
- def prepare_temp_directory
  *Prepare the temporary directory, by default does nothing.*
- def energy_force_driver_all_external_
- def energy_force_driver_all

#### Public Attributes

- trajfnm
  *Name of the trajectory, we need this BEFORE initializing the SuperClass.*
- all_at_once
  *all_at_once is not implemented.*

### 8.2.1 Detailed Description

Subclass of Target for force and energy matching using AMBER.

Implements the prepare and energy_force_driver methods. The get method is in the base class.

Definition at line 168 of file amberio.py.

### 8.2.2 Constructor & Destructor Documentation

#### 8.2.2.1 def forcebalance.amberio.AbInitio_AMBER.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization; define a few core concepts.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 171 of file amberio.py.

### 8.2.3 Member Function Documentation

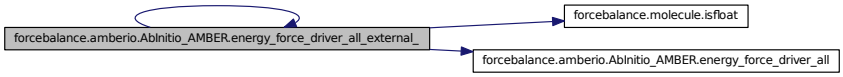
#### 8.2.3.1 def forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all ( *self* )

Definition at line 225 of file amberio.py.

#### 8.2.3.2 def forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all_external_ ( *self* )

Definition at line 187 of file amberio.py.

Here is the call graph for this function:



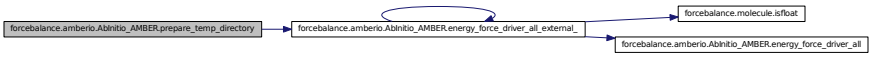

#### 8.2.3.3 def forcebalance.amberio.AbInitio_AMBER.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 178 of file amberio.py.

Here is the call graph for this function:




### 8.2.4 Member Data Documentation

#### 8.2.4.1 forcebalance::amberio.AbInitio_AMBER::all_at_once

all_at_once is not implemented.

Definition at line 173 of file amberio.py.

#### 8.2.4.2 forcebalance::amberio.AbInitio_AMBER::trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 172 of file amberio.py.

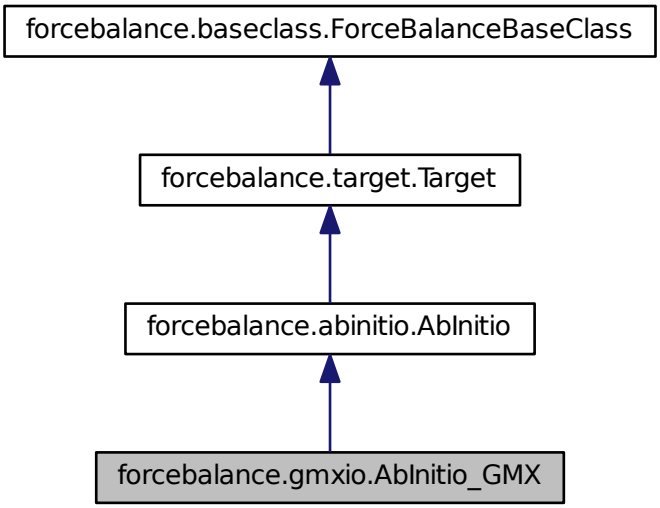
The documentation for this class was generated from the following file:

- amberio.py

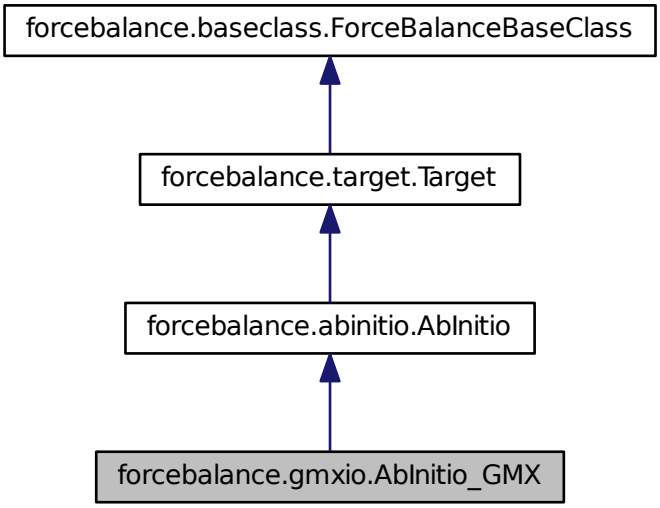
## 8.3 forcebalance.gmxio.AbInitio_GMX Class Reference

Subclass of AbInitio for force and energy matching using normal GROMACS.

Inheritance diagram for forcebalance.gmxio.AbInitio_GMX:




Collaboration diagram for forcebalance.gmxio.AbInitio_GMX:



#### Public Member Functions

- def __init__
  *Initialization; define a few core concepts.*
- def read_topology
  *This function parses the GROMACS topology file (.top) which contains a listing of the molecules in the simulation.*
- def prepare_temp_directory
  *Prepare the temporary directory, by default does nothing.*
- def energy_force_driver
  *Computes the energy and force using GROMACS for a single snapshot.*
- def energy_force_driver_all
  *Computes the energy and force using GROMACS for a trajectory.*
- def generate_vsite_positions
  *Call mdrun in order to update the virtual site positions.*

#### Public Attributes

- trajfnm
  *Name of the trajectory.*
- topfnm
- AtomMask
- topology_flag

### 8.3.1 Detailed Description

Subclass of AbInitio for force and energy matching using normal GROMACS.

Implements the prepare_temp_directory and energy_force_driver methods.

Definition at line 435 of file gmxio.py.

### 8.3.2 Constructor & Destructor Documentation

#### 8.3.2.1 def forcebalance.gmxio.AbInitio_GMX.__init__ ( *self, options, tgt_opts, forcefield* )

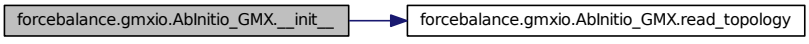
Initialization; define a few core concepts.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 437 of file gmxio.py.

Here is the call graph for this function:



### 8.3.3 Member Function Documentation

#### 8.3.3.1 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver ( *self, shot* )

Computes the energy and force using GROMACS for a single snapshot.

This does not require GROMACS-X2.

Definition at line 515 of file gmxio.py.

Here is the call graph for this function:

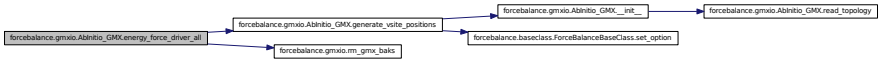

#### 8.3.3.2 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver_all ( *self* )

Computes the energy and force using GROMACS for a trajectory.

This does not require GROMACS-X2.

Definition at line 536 of file gmxio.py.

Here is the call graph for this function:

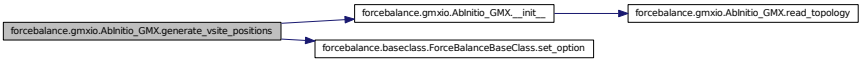



#### 8.3.3.3 def forcebalance.gmxio.AbInitio_GMX.generate_vsite_positions ( *self* )

Call mdrun in order to update the virtual site positions.

Definition at line 559 of file gmxio.py.

Here is the call graph for this function:




#### 8.3.3.4 def forcebalance.gmxio.AbInitio_GMX.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 490 of file gmxio.py.

Here is the call graph for this function:

#### 8.3.3.5 def forcebalance.gmxio.AbInitio_GMX.read_topology ( *self* )

This function parses the GROMACS topology file (.top) which contains a listing of the molecules in the simulation.

For each molecule, it loads up a "FFMolecule" dictionary which contains information about each atom in the molecule - which residue (i.e. molecular fragment) the atom belongs in, the charge group, the particle type etc.

This allows us to do things like condense the gradients into net forces and torques, determine which particles are real atoms and which are virtual sites, and so on.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 452 of file gmxio.py.

### 8.3.4 Member Data Documentation

#### 8.3.4.1 forcebalance::gmxio.AbInitio_GMX::AtomMask

Definition at line 452 of file gmxio.py.

#### 8.3.4.2 forcebalance::gmxio.AbInitio_GMX::topfnm

Definition at line 438 of file gmxio.py.

#### 8.3.4.3 forcebalance::gmxio.AbInitio_GMX::topology_flag

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 452 of file gmxio.py.

#### 8.3.4.4 forcebalance::gmxio.AbInitio_GMX::trajfnm

Name of the trajectory.

Definition at line 438 of file gmxio.py.

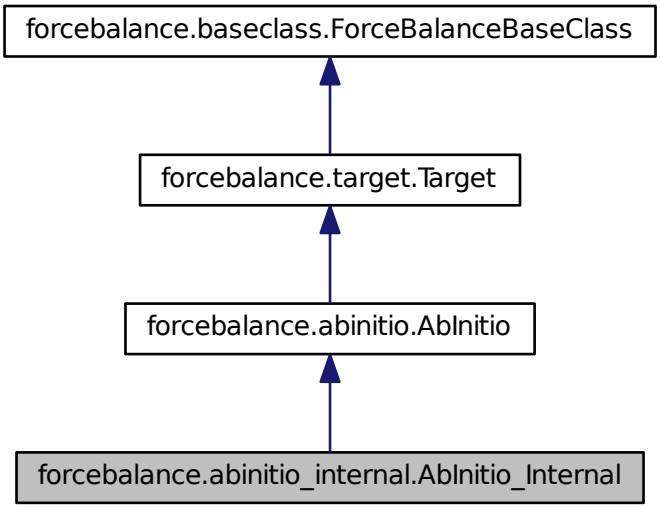
The documentation for this class was generated from the following file:

- gmxio.py

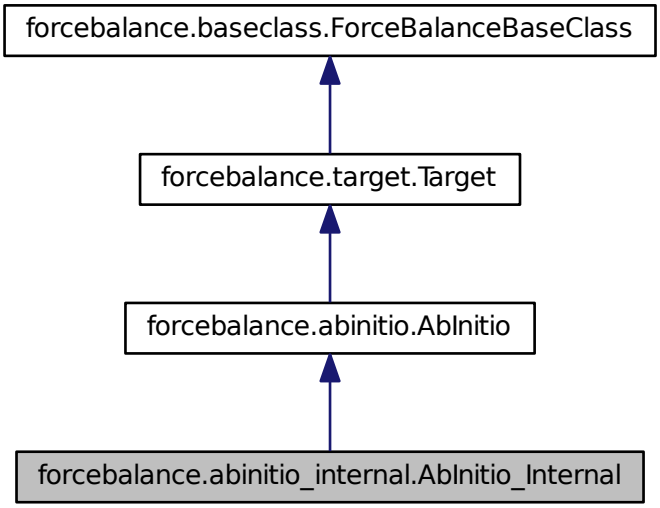
## 8.4 forcebalance.abinitio_internal.AbInitio_Internal Class Reference

Subclass of Target for force and energy matching using an internal implementation.

Inheritance diagram for forcebalance.abinitio_internal.AbInitio_Internal:




Collaboration diagram for forcebalance.abinitio_internal.AbInitio_Internal:



#### Public Member Functions

- def __init__

  *Initialization; define a few core concepts.*
- def energy_force_driver_all

  *Here we actually compute the interactions and return the energies and forces.*

#### Public Attributes

- trajfnm

  *Name of the trajectory, we need this BEFORE initializing the SuperClass.*

### 8.4.1 Detailed Description

Subclass of Target for force and energy matching using an internal implementation.

Implements the prepare and energy_force_driver methods. The get method is in the superclass.

The purpose of this class is to provide an extremely simple test case that does not require the user to install any external software. It only runs with one of the included sample test calculations (internal_tip3p), and the objective function is energy matching.

#### Warning

This class is only intended to work with a very specific test case (internal_tip3p). This is because the topology and ordering of the atoms is hard-coded (12 water molecules with 3 atoms each).
This class does energy matching only (no forces)

Definition at line 37 of file abinitio_internal.py.

### 8.4.2 Constructor & Destructor Documentation

#### 8.4.2.1 def forcebalance.abinitio_internal.AbInitio_Internal.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization; define a few core concepts.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 40 of file abinitio_internal.py.

### 8.4.3 Member Function Documentation

#### 8.4.3.1 def forcebalance.abinitio_internal.AbInitio_Internal.energy_force_driver_all ( *self* )

Here we actually compute the interactions and return the energies and forces.

I verified this to give the same answer as GROMACS.

Definition at line 50 of file abinitio_internal.py.

#### 8.4.4 Member Data Documentation

##### 8.4.4.1 forcebalance::abinitio_internal.AbInitio_Internal::trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 41 of file abinitio_internal.py.

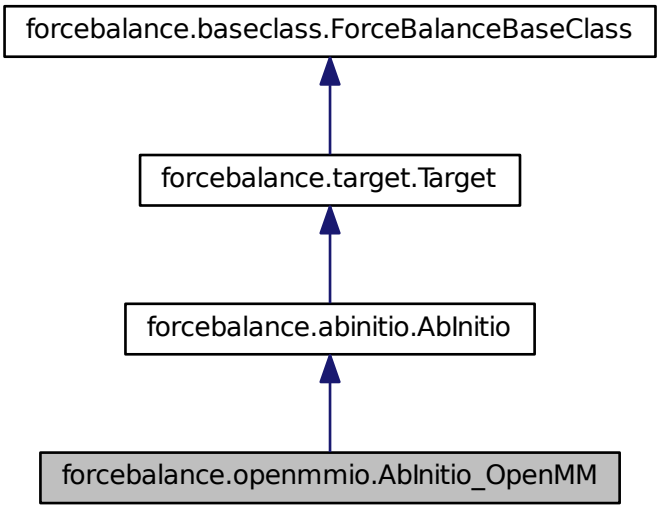
The documentation for this class was generated from the following file:

- abinitio_internal.py

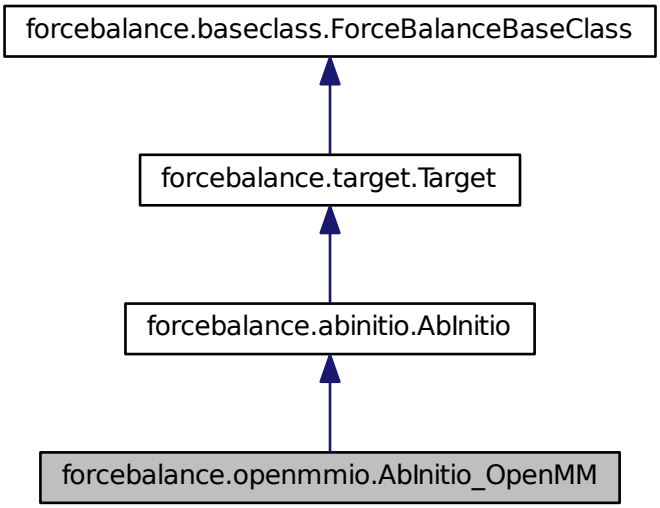
### 8.5 forcebalance.openmmio.AbInitio_OpenMM Class Reference

Subclass of AbInitio for force and energy matching using OpenMM.

Inheritance diagram for forcebalance.openmmio.AbInitio_OpenMM:



Collaboration diagram for forcebalance.openmmio.AbInitio_OpenMM:




**Public Member Functions**

- def __init__

  *Initialization; define a few core concepts.*
- def read_topology
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def energy_force_driver_all_external_
- def energy_force_driver_all_internal_

  *Loop through the snapshots and compute the energies and forces using OpenMM.*
- def energy_force_driver_all

**Public Attributes**

- trajfnm

  *Name of the trajectory, we need this BEFORE initializing the SuperClass.*
- platform

  *Initialize the SuperClass!*
- simulation

  *Create the simulation object within this class itself.*
- xyz_omms
- topology_flag

### 8.5.1 Detailed Description

Subclass of AbInitio for force and energy matching using OpenMM.

Implements the prepare and energy_force_driver methods. The get method is in the superclass.

Definition at line 398 of file openmmio.py.

### 8.5.2 Constructor & Destructor Documentation

#### 8.5.2.1 def forcebalance.openmmio.AbInitio_OpenMM.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization; define a few core concepts.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.abinitio.AbInitio.

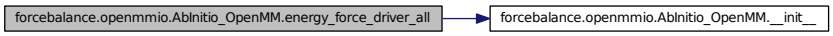
Definition at line 401 of file openmmio.py.

### 8.5.3 Member Function Documentation

#### 8.5.3.1 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all ( *self* )

Definition at line 525 of file openmmio.py.

Here is the call graph for this function:




#### 8.5.3.2 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_external_ ( *self* )

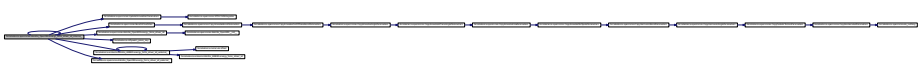
Definition at line 469 of file openmmio.py.

#### 8.5.3.3 def forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_internal_ ( *self* )

Loop through the snapshots and compute the energies and forces using OpenMM.

Definition at line 478 of file openmmio.py.

Here is the call graph for this function:



#### 8.5.3.4 def forcebalance.openmmio.AbInitio_OpenMM.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 464 of file openmmio.py.

Here is the call graph for this function:

#### 8.5.3.5 def forcebalance.openmmio.AbInitio_OpenMM.read_topology ( *self* )

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 454 of file openmmio.py.

### 8.5.4 Member Data Documentation

#### 8.5.4.1 forcebalance::openmmio.AbInitio_OpenMM::platform

Initialize the SuperClass!

Set the device to the environment variable or zero otherwise.

Set the simulation platform

Definition at line 404 of file openmmio.py.

#### 8.5.4.2 forcebalance::openmmio.AbInitio_OpenMM::simulation

Create the simulation object within this class itself.

Definition at line 406 of file openmmio.py.

#### 8.5.4.3 forcebalance::openmmio.AbInitio_OpenMM::topology_flag

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 454 of file openmmio.py.

#### 8.5.4.4 forcebalance::openmmio.AbInitio_OpenMM::trajfnm

Name of the trajectory, we need this BEFORE initializing the SuperClass.

Definition at line 402 of file openmmio.py.

#### 8.5.4.5 forcebalance::openmmio.AbInitio_OpenMM::xyz_omms

Definition at line 406 of file openmmio.py.

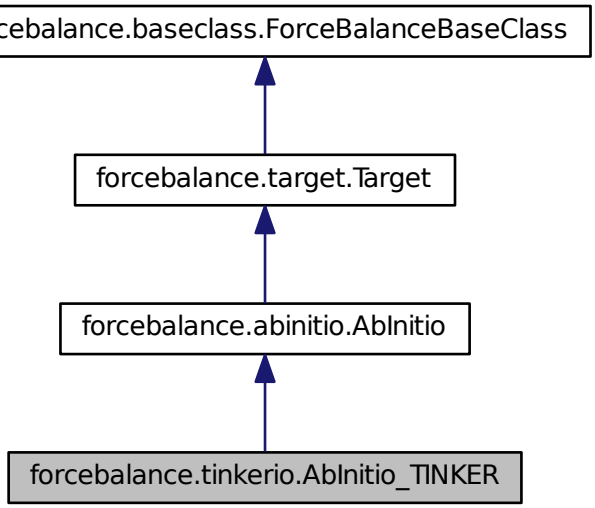
The documentation for this class was generated from the following file:

- openmmio.py

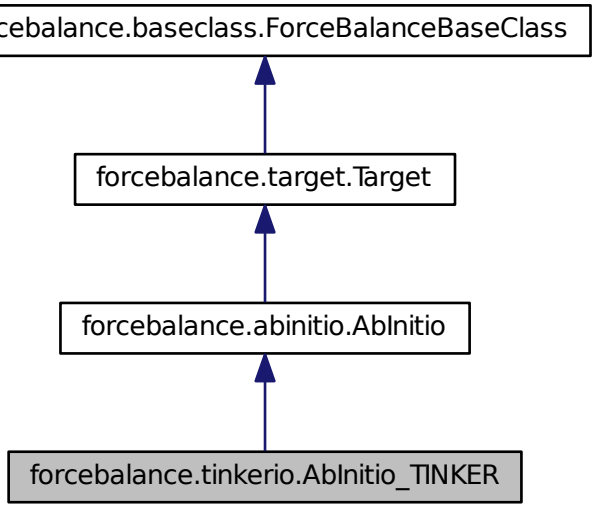
## 8.6 forcebalance.tinkerio.AbInitio_TINKER Class Reference

Subclass of Target for force and energy matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.AbInitio_TINKER:




Collaboration diagram for forcebalance.tinkerio.AbInitio_TINKER:



#### Public Member Functions

- def __init__
  *Initialization; define a few core concepts.*
- def prepare_temp_directory
  *Prepare the temporary directory, by default does nothing.*
- def energy_force_driver
- def energy_driver_all
- def energy_force_driver_all

#### Public Attributes

- trajfnm
  *Name of the trajectory.*
- all_at_once
  *all_at_once is not implemented.*

### 8.6.1 Detailed Description

Subclass of Target for force and energy matching using TINKER.

Implements the prepare and energy_force_driver methods.

Definition at line 294 of file tinkerio.py.

### 8.6.2 Constructor & Destructor Documentation

#### 8.6.2.1 def forcebalance.tinkerio.AbInitio_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization; define a few core concepts.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.abinitio.AbInitio.

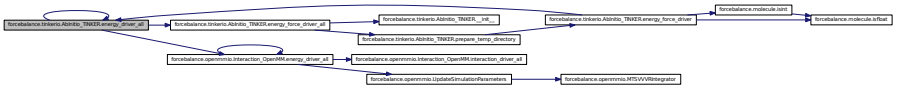
Definition at line 297 of file tinkerio.py.

### 8.6.3 Member Function Documentation

#### 8.6.3.1 def forcebalance.tinkerio.AbInitio_TINKER.energy_driver_all ( *self* )

Definition at line 330 of file tinkerio.py.

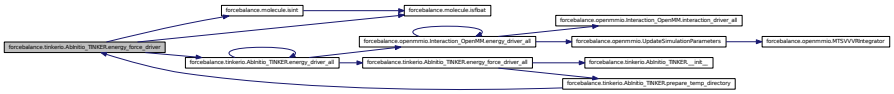
Here is the call graph for this function:



#### 8.6.3.2 def forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver ( *self, shot* )

Definition at line 314 of file tinkerio.py.

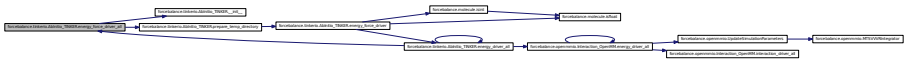
Here is the call graph for this function:



#### 8.6.3.3 def forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver_all ( *self* )

Definition at line 343 of file tinkerio.py.

Here is the call graph for this function:



#### 8.6.3.4 def forcebalance.tinkerio.AbInitio_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.abinitio.AbInitio.

Definition at line 306 of file tinkerio.py.

Here is the call graph for this function:

### 8.6.4 Member Data Documentation

#### 8.6.4.1 forcebalance::tinkerio.AbInitio_TINKER::all_at_once

all_at_once is not implemented.

Definition at line 299 of file tinkerio.py.

#### 8.6.4.2 forcebalance::tinkerio.AbInitio_TINKER::trajfnm

Name of the trajectory.

Definition at line 298 of file tinkerio.py.

The documentation for this class was generated from the following file:

- tinkerio.py

## 8.7 forcebalance.forcefield.BackedUpDict Class Reference

### Public Member Functions

- def __init__
- def __missing__

### Public Attributes

- backup_dict

#### 8.7.1 Detailed Description

Definition at line 173 of file forcefield.py.

#### 8.7.2 Constructor & Destructor Documentation

##### 8.7.2.1 def forcebalance.forcefield.BackedUpDict.__init__ ( *self, backup_dict* )

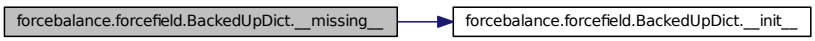
Definition at line 174 of file forcefield.py.

#### 8.7.3 Member Function Documentation

##### 8.7.3.1 def forcebalance.forcefield.BackedUpDict.__missing__ ( *self, key* )

Definition at line 177 of file forcefield.py.

Here is the call graph for this function:




#### 8.7.4 Member Data Documentation

##### 8.7.4.1 forcebalance::forcefield.BackedUpDict::backup_dict

Definition at line 174 of file forcefield.py.

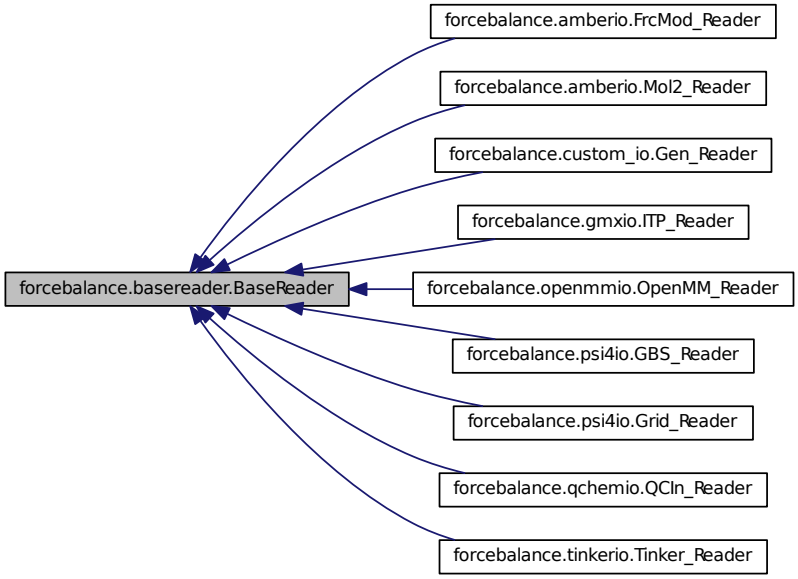
The documentation for this class was generated from the following file:

- forcefield.py

## 8.8 forcebalance.basereader.BaseReader Class Reference

The 'reader' class.

Inheritance diagram for forcebalance.basereader.BaseReader:




### Public Member Functions

- def __init__
- def Split
- def Whites
- def feed
- def build_pid

  *Returns the parameter type (e.g.*

### Public Attributes

- ln
- itype
- suffix
- pdict
- adict

  *The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [ bonds ], [ angles ] etc.*
- molatom

  *The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.*
- Molecules
- AtomTypes

### 8.8.1 Detailed Description

The 'reader' class.

It serves two main functions:

1) When parsing a text force field file, the 'feed' method is called once for every line. Calling the 'feed' method stores the internal variables that are needed for making the unique parameter identifier.

2) The 'reader' also stores the 'pdict' dictionary, which is needed for building the matrix of rescaling factors. This is not needed for the XML force fields, so in XML force fields pdict is replaced with a string called "XML_Override".

Definition at line 25 of file basereader.py.

### 8.8.2 Constructor & Destructor Documentation

#### 8.8.2.1 def forcebalance.basereader.BaseReader.__init__ ( *self, fnm* )

Reimplemented in forcebalance.openmmio.OpenMM_Reader, forcebalance.gmxio.ITP_Reader, forcebalance.psi4io.Grid_Reader, forcebalance.amberio.FrcMod_Reader, forcebalance.tinkerio.Tinker_Reader, forcebalance.custom_io.Gen_Reader, forcebalance.amberio.Mol2_Reader, forcebalance.psi4io.GBS_Reader, and forcebalance.qchemio.QCIn_Reader.

Definition at line 27 of file basereader.py.

### 8.8.3 Member Function Documentation

#### 8.8.3.1 def forcebalance.basereader.BaseReader.build_pid ( *self, pfld* )

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.line_num.field_num'

Reimplemented in forcebalance.psi4io.Grid_Reader, and forcebalance.psi4io.GBS_Reader.

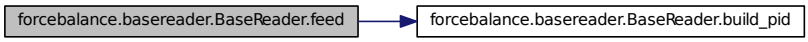
Definition at line 68 of file basereader.py.

#### 8.8.3.2 def forcebalance.basereader.BaseReader.feed ( *self, line* )

Reimplemented in forcebalance.gmxio.ITP_Reader, forcebalance.amberio.FrcMod_Reader, forcebalance.tinkerio.Tinker_Reader, forcebalance.custom_io.Gen_Reader, forcebalance.amberio.Mol2_Reader, and forcebalance.qchemio.QCIn_Reader.

Definition at line 49 of file basereader.py.

Here is the call graph for this function:

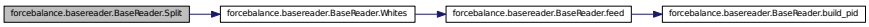



#### 8.8.3.3 def forcebalance.basereader.BaseReader.Split ( *self, line* )

Reimplemented in forcebalance.amberio.FrcMod_Reader.

Definition at line 43 of file basereader.py.

Here is the call graph for this function:




#### 8.8.3.4 def forcebalance.basereader.BaseReader.Whites ( *self, line* )

Reimplemented in forcebalance.amberio.FrcMod_Reader.

Definition at line 46 of file basereader.py.

Here is the call graph for this function:


### 8.8.4 Member Data Documentation

#### 8.8.4.1 forcebalance::basereader.BaseReader::adict

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [ bonds ], [ angles ] etc.

Reimplemented in forcebalance.amberio.FrcMod_Reader, and forcebalance.psi4io.GBS_Reader.

Definition at line 28 of file basereader.py.

#### 8.8.4.2 forcebalance::basereader.BaseReader::AtomTypes

Definition at line 32 of file basereader.py.

#### 8.8.4.3 forcebalance::basereader.BaseReader::itype

Reimplemented in forcebalance.gmxio.ITP_Reader, forcebalance.amberio.FrcMod_Reader, forcebalance.tinkerio.Tinker_Reader, forcebalance.custom_io.Gen_Reader, forcebalance.amberio.Mol2_Reader, and forcebalance.qchemio.QCIn_Reader.

Definition at line 27 of file basereader.py.

#### 8.8.4.4 forcebalance::basereader.BaseReader::ln

Definition at line 27 of file basereader.py.

#### 8.8.4.5 forcebalance::basereader.BaseReader::molatom

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Reimplemented in forcebalance.gmxio.ITP_Reader, and forcebalance.amberio.Mol2_Reader.

Definition at line 32 of file basereader.py.

#### 8.8.4.6 forcebalance::basereader.BaseReader::Molecules

Definition at line 32 of file basereader.py.

#### 8.8.4.7 forcebalance::basereader.BaseReader::pdict

Reimplemented in forcebalance.openmmio.OpenMM_Reader, forcebalance.gmxio.ITP_Reader, forcebalance.amberio.FrcMod_Reader, forcebalance.tinkerio.Tinker_Reader, forcebalance.custom_io.Gen_Reader, forcebalance.amberio.Mol2_Reader, and forcebalance.qchemio.QCIn_Reader.

Definition at line 27 of file basereader.py.

#### 8.8.4.8 forcebalance::basereader.BaseReader::suffix

Reimplemented in forcebalance.gmxio.ITP_Reader, forcebalance.amberio.FrcMod_Reader, forcebalance.tinkerio.Tinker_Reader, forcebalance.custom_io.Gen_Reader, forcebalance.amberio.Mol2_Reader, and forcebalance.qchemio.QCIn_Reader.

Definition at line 27 of file basereader.py.

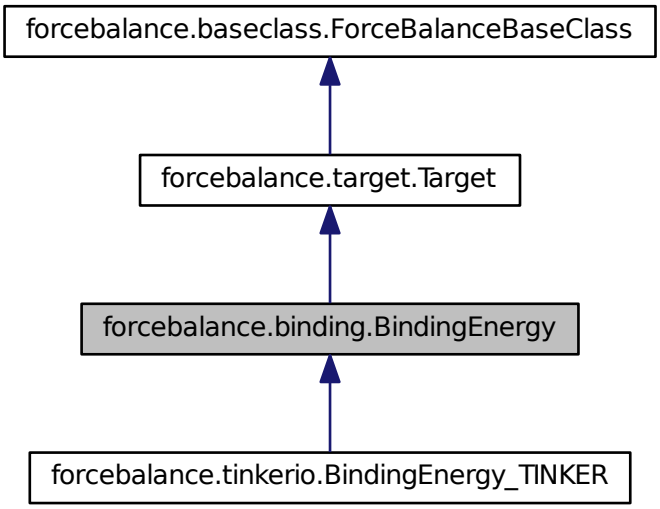
The documentation for this class was generated from the following file:

- basereader.py

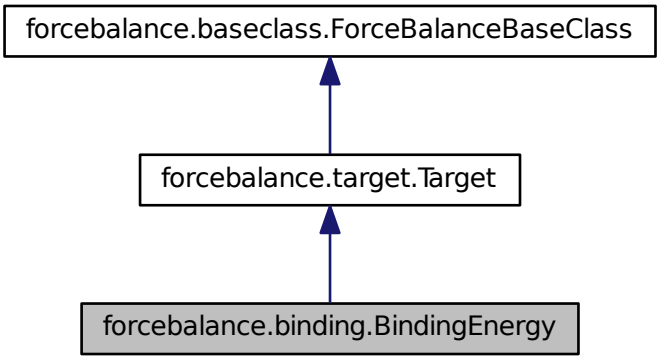
## 8.9 forcebalance.binding.BindingEnergy Class Reference

Improved subclass of Target for fitting force fields to binding energies.

Inheritance diagram for forcebalance.binding.BindingEnergy:




Collaboration diagram for forcebalance.binding.BindingEnergy:




**Public Member Functions**

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*

- def indicate
- def get

  *Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.*

**Public Attributes**

- inter_opts
- PrintDict
- RMSDDict
- rmsd_part
- energy_part
- objective

### 8.9.1 Detailed Description

Improved subclass of Target for fitting force fields to binding energies.

Definition at line 122 of file binding.py.

### 8.9.2 Constructor & Destructor Documentation

#### 8.9.2.1 def forcebalance.binding.BindingEnergy.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.tinkerio.BindingEnergy_TINKER.

Definition at line 125 of file binding.py.

### 8.9.3 Member Function Documentation

#### 8.9.3.1 def forcebalance.binding.BindingEnergy.get ( *self, mvals, AGrad =* `False`*, AHess =* `False` )

Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.

See abinitio for an example.

Reimplemented from forcebalance.target.Target.

Definition at line 161 of file binding.py.

#### 8.9.3.2 def forcebalance.binding.BindingEnergy.indicate ( *self* )

Definition at line 155 of file binding.py.

Here is the call graph for this function:


#### 8.9.4 Member Data Documentation

##### 8.9.4.1 forcebalance::binding.BindingEnergy::energy_part

Definition at line 161 of file binding.py.

##### 8.9.4.2 forcebalance::binding.BindingEnergy::inter_opts

Definition at line 125 of file binding.py.

##### 8.9.4.3 forcebalance::binding.BindingEnergy::objective

Definition at line 161 of file binding.py.

##### 8.9.4.4 forcebalance::binding.BindingEnergy::PrintDict

Definition at line 161 of file binding.py.

##### 8.9.4.5 forcebalance::binding.BindingEnergy::rmsd_part

Definition at line 161 of file binding.py.

##### 8.9.4.6 forcebalance::binding.BindingEnergy::RMSDDict

Definition at line 161 of file binding.py.

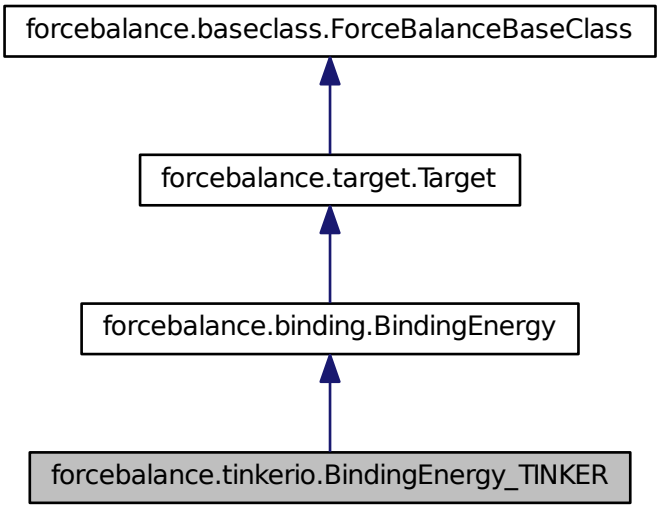
The documentation for this class was generated from the following file:

- binding.py

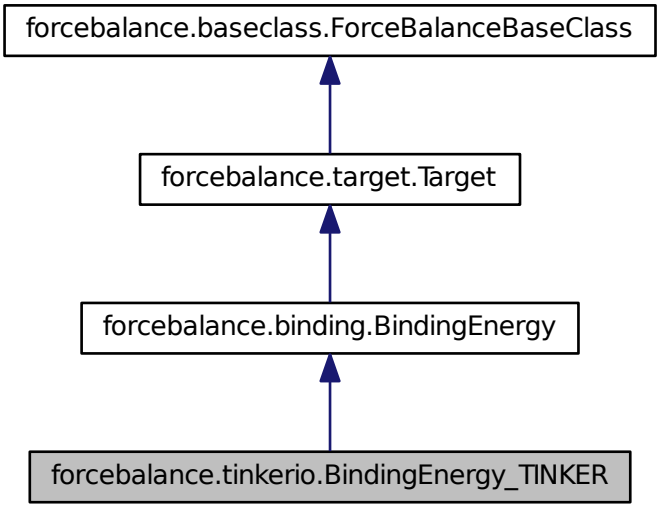
## 8.10 forcebalance.tinkerio.BindingEnergy_TINKER Class Reference

Subclass of BindingEnergy for binding energy matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.BindingEnergy_TINKER:




Collaboration diagram for forcebalance.tinkerio.BindingEnergy_TINKER:



#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def prepare_temp_directory
- def system_driver

#### Public Attributes

- optprog

### 8.10.1 Detailed Description

Subclass of BindingEnergy for binding energy matching using TINKER.

Definition at line 483 of file tinkerio.py.

### 8.10.2 Constructor & Destructor Documentation

#### 8.10.2.1 def forcebalance.tinkerio.BindingEnergy_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.binding.BindingEnergy.

Definition at line 486 of file tinkerio.py.

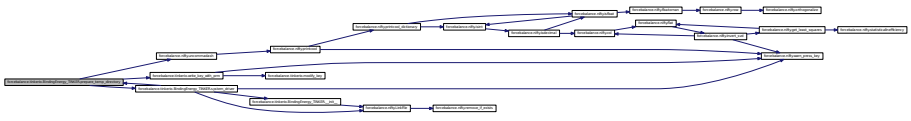
Here is the call graph for this function:


### 8.10.3 Member Function Documentation

#### 8.10.3.1 def forcebalance.tinkerio.BindingEnergy_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

Definition at line 490 of file tinkerio.py.

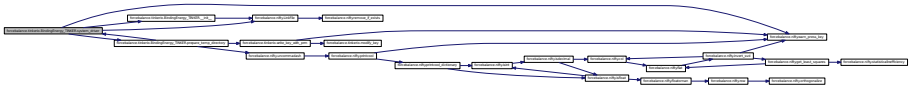
Here is the call graph for this function:



#### 8.10.3.2 def forcebalance.tinkerio.BindingEnergy_TINKER.system_driver ( *self, sysname* )

Definition at line 544 of file tinkerio.py.

Here is the call graph for this function:



### 8.10.4 Member Data Documentation

#### 8.10.4.1 forcebalance::tinkerio.BindingEnergy_TINKER::optprog

Definition at line 490 of file tinkerio.py.

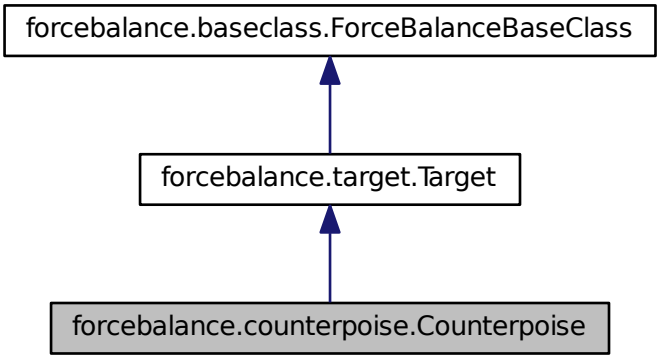
The documentation for this class was generated from the following file:

- tinkerio.py

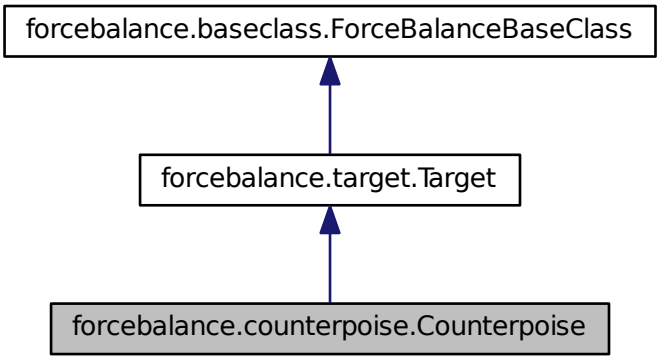
## 8.11 forcebalance.counterpoise.Counterpoise Class Reference

Target subclass for matching the counterpoise correction.

Inheritance diagram for forcebalance.counterpoise.Counterpoise:



Collaboration diagram for forcebalance.counterpoise.Counterpoise:




#### Public Member Functions

- def __init__

  *To instantiate Counterpoise, we read the coordinates and counterpoise data.*
- def loadxyz

  *Parse an XYZ file which contains several xyz coordinates, and return their elements.*
- def load_cp

  *Load in the counterpoise data, which is easy; the file consists of floating point numbers separated by newlines.*
- def get

  *Gets the objective function for fitting the counterpoise correction.*

#### Public Attributes

- xyzs

  *Number of snapshots.*
- cpqm

  *Counterpoise correction data.*
- na

  *Number of atoms.*
- ns

### 8.11.1 Detailed Description

Target subclass for matching the counterpoise correction.

Definition at line 31 of file counterpoise.py.

### 8.11.2 Constructor & Destructor Documentation

#### 8.11.2.1 def forcebalance.counterpoise.Counterpoise.__init__ ( *self, options, tgt_opts, forcefield* )

To instantiate Counterpoise, we read the coordinates and counterpoise data.

Reimplemented from forcebalance.target.Target.

Definition at line 35 of file counterpoise.py.

### 8.11.3 Member Function Documentation

#### 8.11.3.1 def forcebalance.counterpoise.Counterpoise.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Gets the objective function for fitting the counterpoise correction.

As opposed to AbInitio_GMXX2, which calls an external program, this script actually computes the empirical interaction given the force field parameters.

It loops through the snapshots and atom pairs, and computes pairwise contributions to an energy term according to hard-coded functional forms.

One potential issue is that we go through all atom pairs instead of looking only at atom pairs between different fragments. This means that even for two infinitely separated fragments it will predict a finite CP correction. While it might be okay to apply such a potential in practice, there will be some issues for the fitting. Thus, we assume the last snapshot to be CP-free and subtract that value of the potential back out.

Note that forces and parametric derivatives are not implemented.

**Parameters**

| | | |
|---|---|---|
| `in` | *mvals* | Mathematical parameter values |
| `in` | *AGrad* | Switch to turn on analytic gradient (not implemented) |
| `in` | *AHess* | Switch to turn on analytic Hessian (not implemented) |

**Returns**

Answer Contribution to the objective function

Reimplemented from forcebalance.target.Target.

Definition at line 122 of file counterpoise.py.

#### 8.11.3.2 def forcebalance.counterpoise.Counterpoise.load_cp ( *self, fnm* )

Load in the counterpoise data, which is easy; the file consists of floating point numbers separated by newlines.

Definition at line 93 of file counterpoise.py.

#### 8.11.3.3 def forcebalance.counterpoise.Counterpoise.loadxyz ( *self, fnm* )

Parse an XYZ file which contains several xyz coordinates, and return their elements.

**Parameters**

| | | |
|---|---|---|
| `in` | *fnm* | The input XYZ file name |

**Returns**

elem A list of chemical elements in the XYZ file
xyzs A list of XYZ coordinates (number of snapshots times number of atoms)

**Todo** I should probably put this into a more general library for reading coordinates.

Definition at line 61 of file counterpoise.py.

#### 8.11.4 Member Data Documentation

##### 8.11.4.1 forcebalance::counterpoise.Counterpoise::cpqm

Counterpoise correction data.

Definition at line 38 of file counterpoise.py.

##### 8.11.4.2 forcebalance::counterpoise.Counterpoise::na

Number of atoms.

Definition at line 62 of file counterpoise.py.

##### 8.11.4.3 forcebalance::counterpoise.Counterpoise::ns

Definition at line 62 of file counterpoise.py.

##### 8.11.4.4 forcebalance::counterpoise.Counterpoise::xyzs

Number of snapshots.

XYZ elements and coordinates

Definition at line 37 of file counterpoise.py.

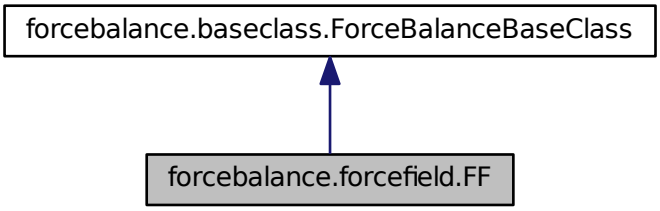
The documentation for this class was generated from the following file:

- counterpoise.py

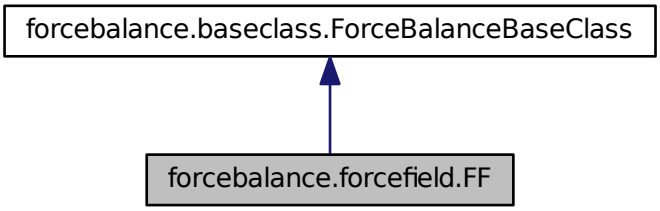
### 8.12 forcebalance.forcefield.FF Class Reference

Force field class.

Inheritance diagram for forcebalance.forcefield.FF:




Collaboration diagram for forcebalance.forcefield.FF:




**Public Member Functions**

- def __init__

  *Instantiation of force field class.*
- def addff

  *Parse a force field file and add it to the class.*
- def addff_txt

  *Parse a text force field and create several important instance variables.*
- def addff_xml

  *Parse an XML force field file and create important instance variables.*
- def make

  *Create a new force field using provided parameter values.*
- def make_redirect
- def find_spacings
- def create_pvals

  *Converts mathematical to physical parameters.*
- def create_mvals

*Converts physical to mathematical parameters.*

- def rsmake

  *Create the rescaling factors for the coordinate transformation in parameter space.*
- def mktransmat

  *Create the transformation matrix to rescale and rotate the mathematical parameters.*
- def list_map

  *Create the plist, which is like a reversed version of the parameter map.*
- def print_map

  *Prints out the (physical or mathematical) parameter indices, IDs and values in a visually appealing way.*
- def assign_p0

  *Assign physical parameter values to the 'pvals0' array.*
- def assign_field

  *Record the locations of a parameter in a txt file; [[file name, line number, field number, and multiplier]].*
- def __eq__

**Public Attributes**

- ffdata

  *As these options proliferate, the force field class becomes less standalone.*
- ffdata_isxml
- map

  *The mapping of interaction type -> parameter number.*
- plist

  *The listing of parameter number -> interaction types.*
- patoms

  *A listing of parameter number -> atoms involved.*
- pfields

  *A list where pfields[pnum] = ['file',line,field,mult,cmd], basically a new way to modify force field files; when we modify the force field file, we go to the specific line/field in a given file and change the number.*
- rs

  *List of rescaling factors.*
- tm

  *The transformation matrix for mathematical -> physical parameters.*
- tmI

  *The transpose of the transformation matrix.*
- excision

  *Indices to exclude from optimization / Hessian inversion.*
- np

  *The total number of parameters.*
- pvals0

  *Initial value of physical parameters.*
- Readers

  *A dictionary of force field reader classes.*
- atomnames

  *A list of atom names (this is new, for ESP fitting)*
- FFAtomTypes

  *WORK IN PROGRESS ## This is a dictionary of {'AtomType':{'Mass' : float, 'Charge' : float, 'ParticleType' : string ('A', 'S', or 'D'), 'AtomicNumber' : int}}.*

- FFMolecules
- redirect

  *Creates plist from map.*
- linedestroy_save

  *Destruction dictionary (experimental).*
- parmdestroy_save
- linedestroy_this
- parmdestroy_this
- tinkerprm
- openmmxml
- qmap
- qid
- qid2

#### 8.12.1 Detailed Description

Force field class.

This class contains all methods for force field manipulation. To create an instance of this class, an input file is required containing the list of force field file names. Everything else inside this class pertaining to force field generation is self-contained.

For details on force field parsing, see the detailed documentation for addff.

Definition at line 194 of file forcefield.py.

#### 8.12.2 Constructor & Destructor Documentation

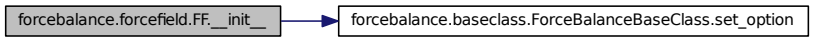
##### 8.12.2.1 def forcebalance.forcefield.FF.__init__ ( *self, options, verbose* = `True` )

Instantiation of force field class.

Many variables here are initialized to zero, but they are filled out by methods like addff, rsmake, and mktransmat.

Definition at line 202 of file forcefield.py.

Here is the call graph for this function:




#### 8.12.3 Member Function Documentation

##### 8.12.3.1 def forcebalance.forcefield.FF.__eq__ ( *self, other* )

Definition at line 1144 of file forcefield.py.

#### 8.12.3.2 def forcebalance.forcefield.FF.addff ( *self, ffname* )

Parse a force field file and add it to the class.

First, figure out the type of force field file. This is done either by explicitly specifying the type using for example, `ffname force_field.xml:openmm` or we figure it out by looking at the file extension.

Next, parse the file. Currently we support two classes of files - text and XML. The two types are treated very differently; for XML we use the parsers in libxml (via the python lxml module), and for text files we have our own in-house parsing class. Within text files, there is also a specialized GROMACS and TINKER parser as well as a generic text parser.

The job of the parser is to determine the following things: 1) Read the user-specified selection of parameters being fitted 2) Build a mapping (dictionary) of `parameter identifier -> index in parameter vector` 3) Build a list of physical parameter values 4) Figure out where to replace the parameter values in the force field file when the values are changed 5) Figure out which parameters need to be repeated or sign-flipped

Generally speaking, each parameter value in the force field file has a `unique parameter identifier` . The identifier consists of three parts - the interaction type, the parameter subtype (within that interaction type), and the atoms involved.

--- If XML: ---

The force field file is read in using the lxml Python module. Specify which parameter you want to fit using by adding a 'parameterize' element to the end of the force field XML file, like so.

```
<AmoebaVdwForce type="BUFFERED-14-7">
   <Vdw class="74" sigma="0.2655" epsilon="0.056484" reduction="0.910"
parameterize="sigma, epsilon, reduction" />
```

In this example, the parameter identifier would look like Vdw/74/epsilon .

--- If GROMACS (.itp) or TINKER (.prm) : ---

Follow the rules in the ITP_Reader or Tinker_Reader derived class. Read the documentation in the class documentation or the 'feed' method to learn more. In all cases the parameter is tagged using # PARM 3 (where # denotes a comment, the word PARM stays the same, and 3 is the field number starting from zero.)

--- If normal text : ---

The parameter identifier is simply built using the file name, line number, and field. Thus, the identifier is unique but completely noninformative (which is not ideal for our purposes, but it works.)

--- Endif ---

**Warning**

My program currently assumes that we are only using one MM program per job. If we use CHARMM and GROMACS to perform simulations as part of the same TARGET, we will get messed up. Maybe this needs to be fixed in the future, with program prefixes to parameters like C_ , G_ .. or simply unit conversions, you get the idea.
I don't think the multiplier actually works for analytic derivatives unless the interaction calculator knows the multiplier as well. I'm sure I can make this work in the future if necessary.

**Parameters**

| | | |
|---|---|---|
| in | *ffname* | Name of the force field file |

Definition at line 393 of file forcefield.py.

#### 8.12.3.3 def forcebalance.forcefield.FF.addff_txt ( *self, ffname, fftype* )

Parse a text force field and create several important instance variables.

Each line is processed using the 'feed' method as implemented in the reader class. This essentially allows us to create the correct parameter identifier (pid), because the pid comes from more than the current line, it also depends on the section that we're in.

When 'PARM' or 'RPT' is encountered, we do several things:

- Build the parameter identifier and insert it into the map
- Point to the file name, line number, and field where the parameter may be modified

Additionally, when 'PARM' is encountered:

- Store the physical parameter value (this is permanent; it's the original value)
- Increment the total number of parameters

When 'RPT' is encountered we don't expand the parameter vector because this parameter is a copy of an existing one. If the parameter identifier is preceded by MINUS_, we chop off the prefix but remember that the sign needs to be flipped.

Definition at line 463 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.4 def forcebalance.forcefield.FF.addff_xml ( *self, ffname* )

Parse an XML force field file and create important instance variables.

This was modeled after addff_txt, but XML and text files are fundamentally different, necessitating two different methods.

We begin with an _ElementTree object. We search through the tree for the 'parameterize' and 'parameter_repeat' keywords. Each time the keyword is encountered, we do the same four actions that I describe in addff_txt.

It's hard to specify precisely the location in an XML file to change a force field parameter. I can create a list of tree elements (essentially pointers to elements within a tree), but this method breaks down when I copy the tree because I have no way to refer to the copied tree elements. Fortunately, lxml gives me a way to represent a tree using a flat list, and my XML file 'locations' are represented using the positions in the list.

**Warning**

The sign-flip hasn't been implemented yet. This shouldn't matter unless your calculation contains repeated parameters with opposite sign.

Definition at line 571 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.5 def forcebalance.forcefield.FF.assign_field ( *self, idx, fnm, ln, pfld, mult, cmd* = None )

Record the locations of a parameter in a txt file; [[file name, line number, field number, and multiplier]].

Note that parameters can have multiple locations because of the repetition functionality.

**Parameters**

| | | |
|---|---|---|
| in | *idx* | The index of the parameter. |
| in | *fnm* | The file name of the parameter field. |
| in | *ln* | The line number within the file (or the node index in the flattened xml) |
| in | *pfld* | The field within the line (or the name of the attribute in the xml) |
| in | *mult* | The multiplier (this is usually 1.0) |

Definition at line 1138 of file forcefield.py.

Here is the call graph for this function:


#### 8.12.3.6 def forcebalance.forcefield.FF.assign_p0 ( *self, idx, val* )

Assign physical parameter values to the 'pvals0' array.

**Parameters**

| | | |
|---|---|---|
| in | *idx* | The index to which we assign the parameter value. |
| in | *val* | The parameter value to be inserted. |

Definition at line 1120 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.7 def forcebalance.forcefield.FF.create_mvals ( *self, pvals* )

Converts physical to mathematical parameters.

We create the inverse transformation matrix using SVD.

**Parameters**

| | | |
|---|---|---|
| in | *pvals* | The physical parameters |

**Returns**

mvals The mathematical parameters

Definition at line 854 of file forcefield.py.

Here is the call graph for this function:


#### 8.12.3.8 def forcebalance.forcefield.FF.create_pvals ( *self, mvals* )

Converts mathematical to physical parameters.

First, mathematical parameters are rescaled and rotated by multiplying by the transformation matrix, followed by adding the original physical parameters.

**Parameters**

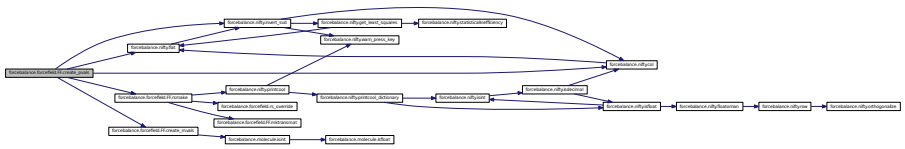
| | | |
|---|---|---|
| in | *mvals* | The mathematical parameters |

**Returns**

pvals The physical parameters

Definition at line 817 of file forcefield.py.

Here is the call graph for this function:



#### 8.12.3.9 def forcebalance.forcefield.FF.find_spacings ( *self* )

Definition at line 772 of file forcefield.py.

Here is the call graph for this function:

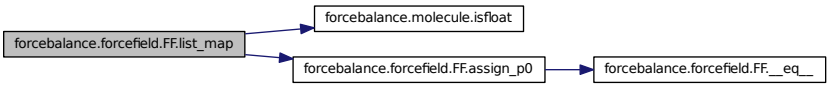
#### 8.12.3.10 def forcebalance.forcefield.FF.list_map ( *self* )

Create the plist, which is like a reversed version of the parameter map.

More convenient for printing.

Definition at line 1097 of file forcefield.py.

Here is the call graph for this function:




#### 8.12.3.11 def forcebalance.forcefield.FF.make ( *self, vals, use_pvals* = `False`, *printdir* = `None`, *precision* = `12` )

Create a new force field using provided parameter values.

This big kahuna does a number of things: 1) Creates the physical parameters from the mathematical parameters 2) Creates force fields with physical parameters substituted in 3) Prints the force fields to the specified file.

It does NOT store the mathematical parameters in the class state (since we can only hold one set of parameters).

**Parameters**

| | | |
|---|---|---|
| `in` | *printdir* | The directory that the force fields are printed to; as usual this is relative to the project root directory. |
| `in` | *vals* | Input parameters. I previously had an option where it uses stored values in the class state, but I don't think that's a good idea anymore. |
| `in` | *use_pvals* | Switch for whether to bypass the coordinate transformation and use physical parameters directly. |

Definition at line 619 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.12 def forcebalance.forcefield.FF.make_redirect ( *self, mvals* )

Definition at line 731 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.13 def forcebalance.forcefield.FF.mktransmat ( *self* )

Create the transformation matrix to rescale and rotate the mathematical parameters.

For point charge parameters, project out perturbations that change the total charge.

First build these:

'qmap' : Just a list of parameter indices that point to charges.

'qid' : For each parameter in the qmap, a list of the affected atoms :) A potential target for the molecule-specific thang.

Then make this:

'qtrans2' : A transformation matrix that rotates the charge parameters. The first row is all zeros (because it corresponds to increasing the charge on all atoms) The other rows correspond to changing one of the parameters and decreasing all of the others equally such that the overall charge is preserved.

'qmat2' : An identity matrix with 'qtrans2' pasted into the right place

'transmat': 'qmat2' with rows and columns scaled using self.rs

'excision': Parameter indices that need to be 'cut out' because they are irrelevant and mess with the matrix diagonalization

**Todo** Only project out changes in total charge of a molecule, and perhaps generalize to fragments of molecules or other types of parameters.

The AMOEBA selection of charge depends not only on the atom type, but what that atom is bonded to.

Definition at line 947 of file forcefield.py.

#### 8.12.3.14 def forcebalance.forcefield.FF.print_map ( *self, vals* = None, *precision* = 4 )

Prints out the (physical or mathematical) parameter indices, IDs and values in a visually appealing way.

Definition at line 1109 of file forcefield.py.

Here is the call graph for this function:

#### 8.12.3.15 def forcebalance.forcefield.FF.rsmake ( *self, printfacs* = `True` )

Create the rescaling factors for the coordinate transformation in parameter space.

The proper choice of rescaling factors (read: prior widths in maximum likelihood analysis) is still a black art. This is a topic of current research.

**Todo** Pass in rsfactors through the input file

**Parameters**

| | | |
|---|---|---|
| `in` | *printfacs* | List for printing out the resecaling factors |

Definition at line 871 of file forcefield.py.

Here is the call graph for this function:

### 8.12.4 Member Data Documentation

#### 8.12.4.1 forcebalance::forcefield.FF::atomnames

A list of atom names (this is new, for ESP fitting)

Definition at line 229 of file forcefield.py.

#### 8.12.4.2 forcebalance::forcefield.FF::excision

Indices to exclude from optimization / Hessian inversion.

Some customized constraints here.

Quadrupoles must be traceless

Definition at line 225 of file forcefield.py.

#### 8.12.4.3 forcebalance::forcefield.FF::FFAtomTypes

WORK IN PROGRESS ## This is a dictionary of {'AtomType':{'Mass' : float, 'Charge' : float, 'ParticleType' : string ('A', 'S', or 'D'), 'AtomicNumber' : int}}.

Definition at line 231 of file forcefield.py.

#### 8.12.4.4 forcebalance::forcefield.FF::ffdata

As these options proliferate, the force field class becomes less standalone.

I need to think of a good solution here... The root directory of the project File names of force fields Directory containing force fields, relative to project directory Priors given by the user :) Whether to constrain the charges. Whether to constrain the charges. Switch for AMOEBA direct or mutual. Switch for rigid water molecules Bypass the transformation and use physical parameters directly The content of all force field files are stored in memory

Definition at line 214 of file forcefield.py.

#### 8.12.4.5 forcebalance::forcefield.FF::ffdata_isxml

Definition at line 214 of file forcefield.py.

#### 8.12.4.6 forcebalance::forcefield.FF::FFMolecules

Definition at line 231 of file forcefield.py.

#### 8.12.4.7 forcebalance::forcefield.FF::linedestroy_save

Destruction dictionary (experimental).

Definition at line 238 of file forcefield.py.

#### 8.12.4.8 forcebalance::forcefield.FF::linedestroy_this

Definition at line 238 of file forcefield.py.

#### 8.12.4.9 forcebalance::forcefield.FF::map

The mapping of interaction type -> parameter number.

Definition at line 215 of file forcefield.py.

#### 8.12.4.10 forcebalance::forcefield.FF::np

The total number of parameters.

Definition at line 226 of file forcefield.py.

#### 8.12.4.11 forcebalance::forcefield.FF::openmmxml

Definition at line 393 of file forcefield.py.

#### 8.12.4.12 forcebalance::forcefield.FF::parmdestroy_save

Definition at line 238 of file forcefield.py.

#### 8.12.4.13 forcebalance::forcefield.FF::parmdestroy_this

Definition at line 238 of file forcefield.py.

#### 8.12.4.14 forcebalance::forcefield.FF::patoms

A listing of parameter number -> atoms involved.

Definition at line 217 of file forcefield.py.

#### 8.12.4.15 forcebalance::forcefield.FF::pfields

A list where pfields[pnum] = ['file',line,field,mult,cmd], basically a new way to modify force field files; when we modify the force field file, we go to the specific line/field in a given file and change the number.

Definition at line 221 of file forcefield.py.

#### 8.12.4.16 forcebalance::forcefield.FF::plist

The listing of parameter number -> interaction types.

Definition at line 216 of file forcefield.py.

#### 8.12.4.17 forcebalance::forcefield.FF::pvals0

Initial value of physical parameters.

Definition at line 227 of file forcefield.py.

#### 8.12.4.18 forcebalance::forcefield.FF::qid

Definition at line 947 of file forcefield.py.

#### 8.12.4.19 forcebalance::forcefield.FF::qid2

Definition at line 947 of file forcefield.py.

#### 8.12.4.20 forcebalance::forcefield.FF::qmap

Definition at line 947 of file forcefield.py.

#### 8.12.4.21 forcebalance::forcefield.FF::Readers

A dictionary of force field reader classes.

Definition at line 228 of file forcefield.py.

#### 8.12.4.22 forcebalance::forcefield.FF::redirect

Creates plist from map.

Prints the plist to screen. Make the rescaling factors. Make the transformation matrix. Redirection dictionary (experimental).

Definition at line 237 of file forcefield.py.

#### 8.12.4.23 forcebalance::forcefield.FF::rs

List of rescaling factors.

Takes the dictionary 'BONDS':{3:'B', 4:'K'}, 'VDW':{4:'S', 5:'T'}, and turns it into a list of term types ['BONDSB','BONDSK','VDWS','VDWT'].

The array of rescaling factors

Definition at line 222 of file forcefield.py.

#### 8.12.4.24 forcebalance::forcefield.FF::tinkerprm

Definition at line 393 of file forcefield.py.

#### 8.12.4.25 forcebalance::forcefield.FF::tm

The transformation matrix for mathematical -> physical parameters.

Definition at line 223 of file forcefield.py.

#### 8.12.4.26 forcebalance::forcefield.FF::tml

The transpose of the transformation matrix.

Definition at line 224 of file forcefield.py.

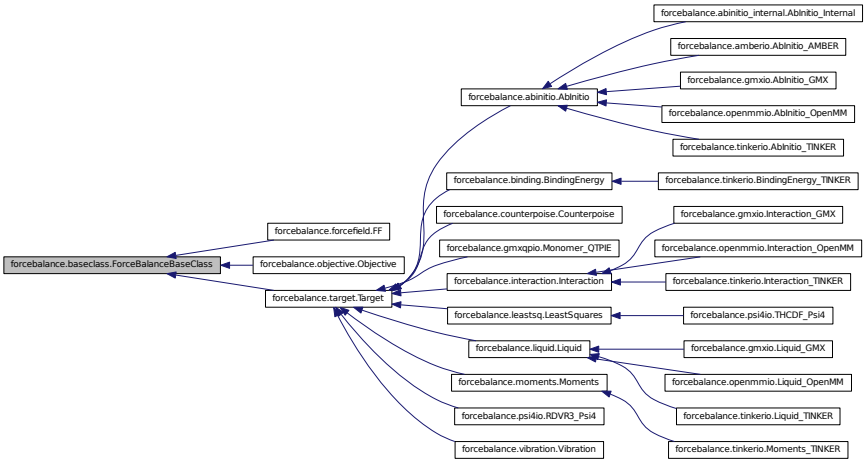
The documentation for this class was generated from the following file:

- forcefield.py

## 8.13 forcebalance.baseclass.ForceBalanceBaseClass Class Reference

Provides some nifty functions that are common to all ForceBalance classes.

Inheritance diagram for forcebalance.baseclass.ForceBalanceBaseClass:




### Public Member Functions

- def __init__
- def set_option

### Public Attributes

- PrintOptionDict
- verbose_options

#### 8.13.1 Detailed Description

Provides some nifty functions that are common to all ForceBalance classes.

Definition at line 6 of file baseclass.py.

#### 8.13.2 Constructor & Destructor Documentation

##### 8.13.2.1 def forcebalance.baseclass.ForceBalanceBaseClass.__init__ ( *self, options* )

Definition at line 8 of file baseclass.py.

#### 8.13.3 Member Function Documentation

##### 8.13.3.1 def forcebalance.baseclass.ForceBalanceBaseClass.set_option ( *self, in_dict, src_key, dest_key* = None, *val* = None, *default* = None, *forceprint* = False )

Definition at line 12 of file baseclass.py.

#### 8.13.4 Member Data Documentation

##### 8.13.4.1 forcebalance::baseclass.ForceBalanceBaseClass::PrintOptionDict

Definition at line 8 of file baseclass.py.

##### 8.13.4.2 forcebalance::baseclass.ForceBalanceBaseClass::verbose_options

Definition at line 8 of file baseclass.py.

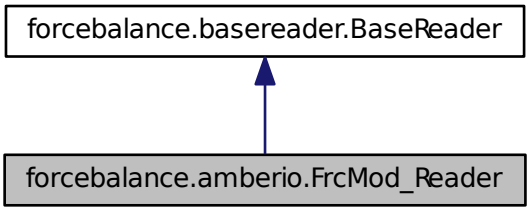
The documentation for this class was generated from the following file:

- baseclass.py

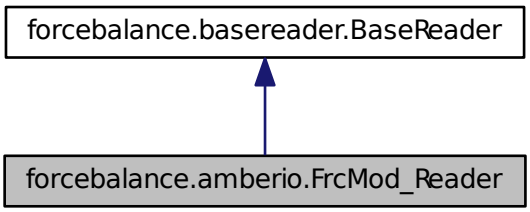
## 8.14 forcebalance.amberio.FrcMod_Reader Class Reference

Finite state machine for parsing FrcMod force field file.

Inheritance diagram for forcebalance.amberio.FrcMod_Reader:



Collaboration diagram for forcebalance.amberio.FrcMod_Reader:




#### Public Member Functions

- def __init__
- def Split
- def Whites
- def feed

#### Public Attributes

- pdict

  *The parameter dictionary (defined in this file)*
- atom

  *The atom numbers in the interaction (stored in the parser)*
- dihe

  *Whether we're inside the dihedral section.*
- adict

  *The frcmod file never has any atoms in it.*
- itype
- suffix

### 8.14.1 Detailed Description

Finite state machine for parsing FrcMod force field file.

Definition at line 96 of file amberio.py.

### 8.14.2 Constructor & Destructor Documentation

#### 8.14.2.1 def forcebalance.amberio.FrcMod_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 98 of file amberio.py.

### 8.14.3 Member Function Documentation

#### 8.14.3.1 def forcebalance.amberio.FrcMod_Reader.feed ( *self, line* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 116 of file amberio.py.

#### 8.14.3.2 def forcebalance.amberio.FrcMod_Reader.Split ( *self, line* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 110 of file amberio.py.

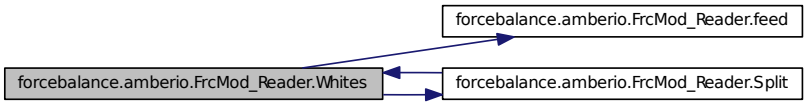
Here is the call graph for this function:


#### 8.14.3.3 def forcebalance.amberio.FrcMod_Reader.Whites ( *self, line* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 113 of file amberio.py.

Here is the call graph for this function:




### 8.14.4 Member Data Documentation

#### 8.14.4.1 forcebalance::amberio.FrcMod_Reader::adict

The frcmod file never has any atoms in it.

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 102 of file amberio.py.

#### 8.14.4.2 forcebalance::amberio.FrcMod_Reader::atom

The atom numbers in the interaction (stored in the parser)

Definition at line 100 of file amberio.py.

#### 8.14.4.3 forcebalance::amberio.FrcMod_Reader::dihe

Whether we're inside the dihedral section.

Definition at line 101 of file amberio.py.

#### 8.14.4.4 forcebalance::amberio.FrcMod_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 116 of file amberio.py.

#### 8.14.4.5 forcebalance::amberio.FrcMod_Reader::pdict

The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 99 of file amberio.py.

#### 8.14.4.6 forcebalance::amberio.FrcMod_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 116 of file amberio.py.

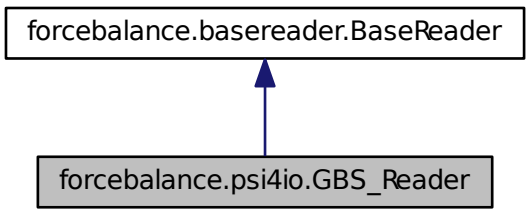
The documentation for this class was generated from the following file:

- amberio.py

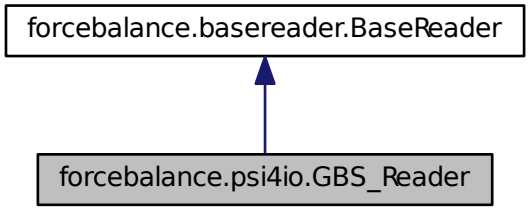
## 8.15 forcebalance.psi4io.GBS_Reader Class Reference

Interaction type -> Parameter Dictionary.

Inheritance diagram for forcebalance.psi4io.GBS_Reader:



Collaboration diagram for forcebalance.psi4io.GBS_Reader:




#### Public Member Functions

- def __init__
- def build_pid

  *Returns the parameter type (e.g.*
- def feed

  *Feed in a line.*

#### Public Attributes

- element
- amom
- last_amom
- basis_number
- contraction_number
- adict

  *The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [ bonds ], [ angles ] etc.*
- isdata
- destroy

### 8.15.1 Detailed Description

Interaction type -> Parameter Dictionary.

pdict = {'Exponent':{0:'A', 1:'C'}, 'BASSP' :{0:'A', 1:'B', 2:'C'} } Finite state machine for parsing basis set files.

Definition at line 32 of file psi4io.py.

### 8.15.2 Constructor & Destructor Documentation

#### 8.15.2.1 def forcebalance.psi4io.GBS_Reader.__init__ ( *self, fnm* = `None` )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 34 of file psi4io.py.

### 8.15.3 Member Function Documentation

#### 8.15.3.1 def forcebalance.psi4io.GBS_Reader.build_pid ( *self, pfld* )

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

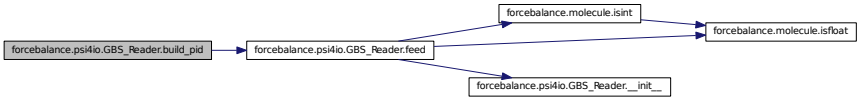
If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 45 of file psi4io.py.

Here is the call graph for this function:




#### 8.15.3.2 def forcebalance.psi4io.GBS_Reader.feed ( *self, line, linindep* = `False` )

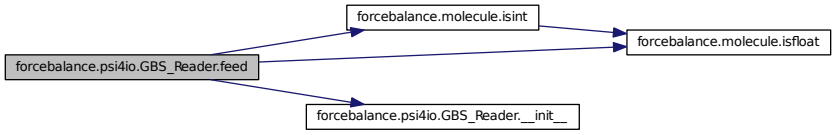
Feed in a line.

**Parameters**

| | | |
|---|---|---|
| `in` | *line* | The line of data |

Definition at line 58 of file psi4io.py.

Here is the call graph for this function:




### 8.15.4 Member Data Documentation

#### 8.15.4.1 forcebalance::psi4io.GBS_Reader::adict

The mapping of (this residue, atom number) to (atom name) for building atom-specific interactions in [ bonds ], [ angles ] etc.

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 34 of file psi4io.py.

#### 8.15.4.2 forcebalance::psi4io.GBS_Reader::amom

Definition at line 34 of file psi4io.py.

#### 8.15.4.3 forcebalance::psi4io.GBS_Reader::basis_number

Definition at line 34 of file psi4io.py.

#### 8.15.4.4 forcebalance::psi4io.GBS_Reader::contraction_number

Definition at line 34 of file psi4io.py.

#### 8.15.4.5 forcebalance::psi4io.GBS_Reader::destroy

Definition at line 34 of file psi4io.py.

#### 8.15.4.6 forcebalance::psi4io.GBS_Reader::element

Definition at line 34 of file psi4io.py.

#### 8.15.4.7 forcebalance::psi4io.GBS_Reader::isdata

Definition at line 34 of file psi4io.py.

#### 8.15.4.8 forcebalance::psi4io.GBS_Reader::last_amom

Definition at line 34 of file psi4io.py.

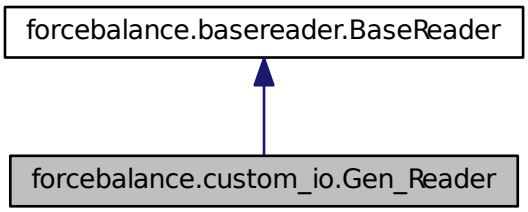
The documentation for this class was generated from the following file:

- psi4io.py

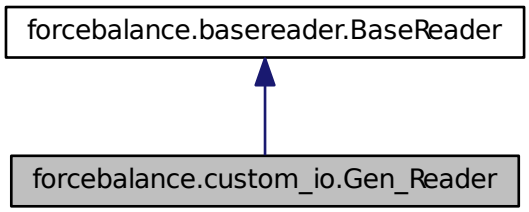
## 8.16 forcebalance.custom_io.Gen_Reader Class Reference

Finite state machine for parsing custom GROMACS force field files.

Inheritance diagram for forcebalance.custom_io.Gen_Reader:



Collaboration diagram for forcebalance.custom_io.Gen_Reader:




#### Public Member Functions

- def __init__
- def feed

  *Feed in a line.*

#### Public Attributes

- sec

  *The current section that we're in.*
- pdict

  *The parameter dictionary (defined in this file)*
- itype
- suffix

### 8.16.1 Detailed Description

Finite state machine for parsing custom GROMACS force field files.

This class is instantiated when we begin to read in a file. The feed(line) method updates the state of the machine, giving it information like the residue we're currently on, the nonbonded interaction type, and the section that we're in. Using this information we can look up the interaction type and parameter type for building the parameter ID.

Definition at line 41 of file custom_io.py.

### 8.16.2 Constructor & Destructor Documentation

#### 8.16.2.1 def forcebalance.custom_io.Gen_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 43 of file custom_io.py.

### 8.16.3 Member Function Documentation

#### 8.16.3.1 def forcebalance.custom_io.Gen_Reader.feed ( *self, line* )

Feed in a line.

**Parameters**

| | | |
|---|---|---|
| in | *line* | The line of data |

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 57 of file custom_io.py.

### 8.16.4 Member Data Documentation

#### 8.16.4.1 forcebalance::custom_io.Gen_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 57 of file custom_io.py.

#### 8.16.4.2 forcebalance::custom_io.Gen_Reader::pdict

The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 45 of file custom_io.py.

#### 8.16.4.3 forcebalance::custom_io.Gen_Reader::sec

The current section that we're in.

Definition at line 44 of file custom_io.py.

#### 8.16.4.4 forcebalance::custom_io.Gen_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 57 of file custom_io.py.

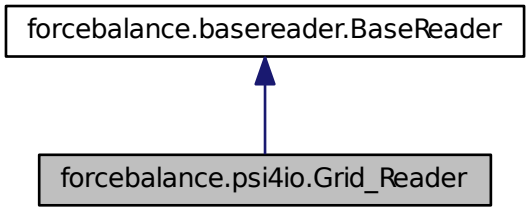
The documentation for this class was generated from the following file:

- custom_io.py

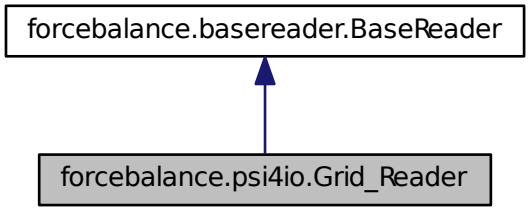
## 8.17 forcebalance.psi4io.Grid_Reader Class Reference

Finite state machine for parsing DVR grid files.

Inheritance diagram for forcebalance.psi4io.Grid_Reader:




Collaboration diagram for forcebalance.psi4io.Grid_Reader:




#### Public Member Functions

- def __init__
- def build_pid

  *Returns the parameter type (e.g.*
- def feed

  *Feed in a line.*

#### Public Attributes

- element
- point
- radii
- isdata

### 8.17.1 Detailed Description

Finite state machine for parsing DVR grid files.

Definition at line 246 of file psi4io.py.

### 8.17.2 Constructor & Destructor Documentation

#### 8.17.2.1 def forcebalance.psi4io.Grid_Reader.__init__ ( *self, fnm* = `None` )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 248 of file psi4io.py.

### 8.17.3 Member Function Documentation

#### 8.17.3.1 def forcebalance.psi4io.Grid_Reader.build_pid ( *self, pfld* )

Returns the parameter type (e.g.

K in BONDSK) based on the current interaction type.

Both the 'pdict' dictionary (see gmxio.pdict) and the interaction type 'state' (here, BONDS) are needed to get the parameter type.

If, however, 'pdict' does not contain the ptype value, a suitable substitute is simply the field number.

Note that if the interaction type state is not set, then it defaults to the file name, so a generic parameter ID is 'filename.-line_num.field_num'

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 254 of file psi4io.py.

Here is the call graph for this function:


#### 8.17.3.2 def forcebalance.psi4io.Grid_Reader.feed ( *self, line, linindep* = `False` )

Feed in a line.

**Parameters**

| | | |
|---|---|---|
| `in` | *line* | The line of data |

Definition at line 269 of file psi4io.py.

### 8.17.4 Member Data Documentation

#### 8.17.4.1 forcebalance::psi4io.Grid_Reader::element

Definition at line 248 of file psi4io.py.

#### 8.17.4.2 forcebalance::psi4io.Grid_Reader::isdata

Definition at line 269 of file psi4io.py.

#### 8.17.4.3 forcebalance::psi4io.Grid_Reader::point

Definition at line 248 of file psi4io.py.

#### 8.17.4.4 forcebalance::psi4io.Grid_Reader::radii

Definition at line 248 of file psi4io.py.

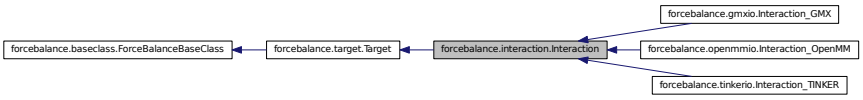
The documentation for this class was generated from the following file:

- psi4io.py

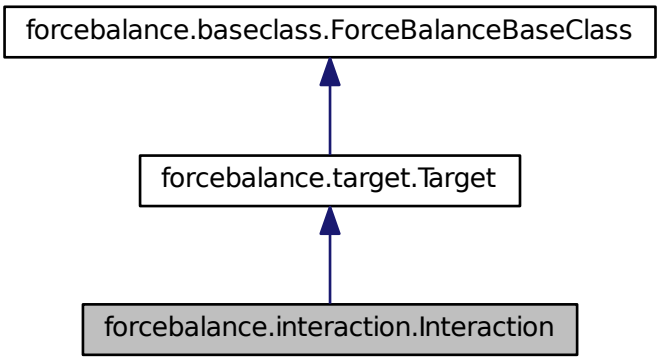
## 8.18 forcebalance.interaction.Interaction Class Reference

Subclass of Target for fitting force fields to interaction energies.

Inheritance diagram for forcebalance.interaction.Interaction:




Collaboration diagram for forcebalance.interaction.Interaction:



#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def read_reference_data

  *Read the reference ab initio data from a file such as qdata.txt.*
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def indicate
- def get

  *Evaluate objective function.*

#### Public Attributes

- select1

  *Number of snapshots.*
- select2

  *Set fragment 2.*
- eqm

  *Set upper cutoff energy.*
- label

  *Snapshot label, useful for graphing.*
- qfnm

  *The qdata.txt file that contains the QM energies and forces.*
- e_err

  *Qualitative Indicator: average energy error (in kJ/mol)*
- e_err_pct
- ns

  *Read in the trajectory file.*
- traj
- divisor

  *Read in the reference data.*
- prefactor
- weight
- emm
- objective

### 8.18.1 Detailed Description

Subclass of Target for fitting force fields to interaction energies.

Currently TINKER is supported.

We introduce the following concepts:

- The number of snapshots
- The reference interaction energies and the file they belong in (qdata.txt)

This subclass contains the 'get' method for building the objective function from any simulation software (a driver to run the program and read output is still required).

Definition at line 32 of file interaction.py.

### 8.18.2 Constructor & Destructor Documentation

#### 8.18.2.1 def forcebalance.interaction.Interaction.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.gmxio.Interaction_GMX, forcebalance.tinkerio.Interaction_TINKER, and forcebalance.openmmio.Interaction_OpenMM.

Definition at line 35 of file interaction.py.

### 8.18.3 Member Function Documentation

#### 8.18.3.1 def forcebalance.interaction.Interaction.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Evaluate objective function.

Reimplemented from forcebalance.target.Target.

Definition at line 160 of file interaction.py.

#### 8.18.3.2 def forcebalance.interaction.Interaction.indicate ( *self* )

Definition at line 144 of file interaction.py.

Here is the call graph for this function:


#### 8.18.3.3 def forcebalance.interaction.Interaction.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented in forcebalance.gmxio.Interaction_GMX, forcebalance.tinkerio.Interaction_TINKER, and forcebalance.openmmio.Interaction_OpenMM.

Definition at line 141 of file interaction.py.

#### 8.18.3.4 def forcebalance.interaction.Interaction.read_reference_data ( *self* )

Read the reference ab initio data from a file such as qdata.txt.

After reading in the information from qdata.txt, it is converted into the GROMACS/OpenMM units (kJ/mol for energy, kJ/mol/nm force).

Definition at line 122 of file interaction.py.

Here is the call graph for this function:


#### 8.18.4 Member Data Documentation

##### 8.18.4.1 forcebalance::interaction.Interaction::divisor

Read in the reference data.

Prepare the temporary directory

Definition at line 50 of file interaction.py.

##### 8.18.4.2 forcebalance::interaction.Interaction::e_err

Qualitative Indicator: average energy error (in kJ/mol)

Definition at line 47 of file interaction.py.

##### 8.18.4.3 forcebalance::interaction.Interaction::e_err_pct

Definition at line 47 of file interaction.py.

##### 8.18.4.4 forcebalance::interaction.Interaction::emm

Definition at line 160 of file interaction.py.

##### 8.18.4.5 forcebalance::interaction.Interaction::eqm

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 44 of file interaction.py.

##### 8.18.4.6 forcebalance::interaction.Interaction::label

Snapshot label, useful for graphing.

Definition at line 45 of file interaction.py.

##### 8.18.4.7 forcebalance::interaction.Interaction::ns

Read in the trajectory file.

Definition at line 48 of file interaction.py.

##### 8.18.4.8 forcebalance::interaction.Interaction::objective

Definition at line 160 of file interaction.py.

#### 8.18.4.9 forcebalance::interaction.Interaction::prefactor

Definition at line 50 of file interaction.py.

#### 8.18.4.10 forcebalance::interaction.Interaction::qfnm

The qdata.txt file that contains the QM energies and forces.

Definition at line 46 of file interaction.py.

#### 8.18.4.11 forcebalance::interaction.Interaction::select1

Number of snapshots.

Do we call Q-Chem for dielectric energies? (Currently needs to be fixed) Do we put the reference energy into the denominator? Do we put the reference energy into the denominator? What is the energy denominator? Set fragment 1

Definition at line 41 of file interaction.py.

#### 8.18.4.12 forcebalance::interaction.Interaction::select2

Set fragment 2.

Definition at line 42 of file interaction.py.

#### 8.18.4.13 forcebalance::interaction.Interaction::traj

Definition at line 48 of file interaction.py.

#### 8.18.4.14 forcebalance::interaction.Interaction::weight

Definition at line 160 of file interaction.py.

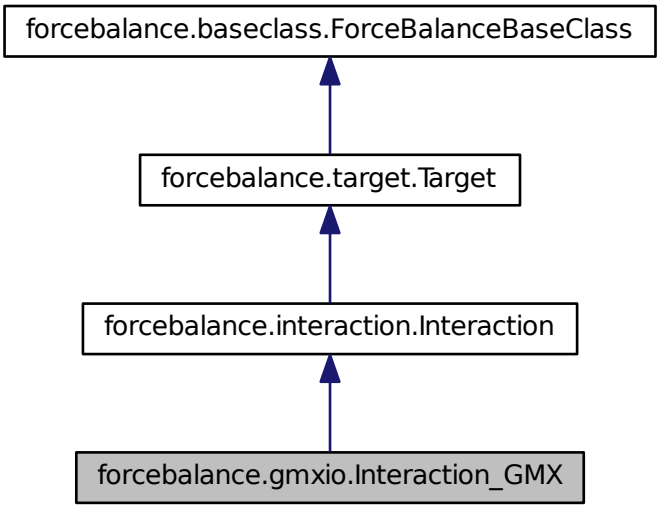
The documentation for this class was generated from the following file:

- interaction.py

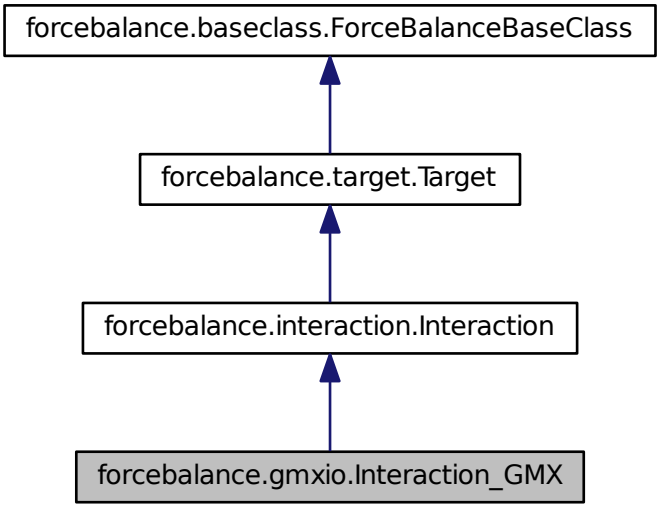
## 8.19 forcebalance.gmxio.Interaction_GMX Class Reference

Subclass of Interaction for interaction energy matching using GROMACS.

Inheritance diagram for forcebalance.gmxio.Interaction_GMX:




Collaboration diagram for forcebalance.gmxio.Interaction_GMX:



#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def interaction_driver

  *Computes the energy and force using GROMACS for a single snapshot.*
- def interaction_driver_all

  *Computes the energy and force using GROMACS for a trajectory.*

#### Public Attributes

- trajfnm

  *Name of the trajectory.*
- topfnm
- Dielectric
- Diel_B

### 8.19.1 Detailed Description

Subclass of Interaction for interaction energy matching using GROMACS.

Definition at line 618 of file gmxio.py.

### 8.19.2 Constructor & Destructor Documentation

#### 8.19.2.1 def forcebalance.gmxio.Interaction_GMX.__init__ ( *self, options, tgt_opts, forcefield* )

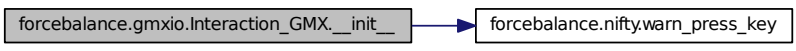
All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 620 of file gmxio.py.

Here is the call graph for this function:



### 8.19.3 Member Function Documentation

#### 8.19.3.1 def forcebalance.gmxio.Interaction_GMX.interaction_driver ( *self, shot* )

Computes the energy and force using GROMACS for a single snapshot.

This does not require GROMACS-X2.

Definition at line 651 of file gmxio.py.

#### 8.19.3.2 def forcebalance.gmxio.Interaction_GMX.interaction_driver_all ( *self, dielectric* = `False` )

Computes the energy and force using GROMACS for a trajectory.

This does not require GROMACS-X2.

Definition at line 656 of file gmxio.py.

Here is the call graph for this function:

#### 8.19.3.3 def forcebalance.gmxio.Interaction_GMX.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 628 of file gmxio.py.

Here is the call graph for this function:

### 8.19.4 Member Data Documentation

#### 8.19.4.1 forcebalance::gmxio.Interaction_GMX::Diel_B

Definition at line 656 of file gmxio.py.

#### 8.19.4.2 forcebalance::gmxio.Interaction_GMX::Dielectric

Definition at line 621 of file gmxio.py.

#### 8.19.4.3 forcebalance::gmxio.Interaction_GMX::topfnm

Definition at line 621 of file gmxio.py.

#### 8.19.4.4 forcebalance::gmxio.Interaction_GMX::trajfnm

Name of the trajectory.

Definition at line 621 of file gmxio.py.

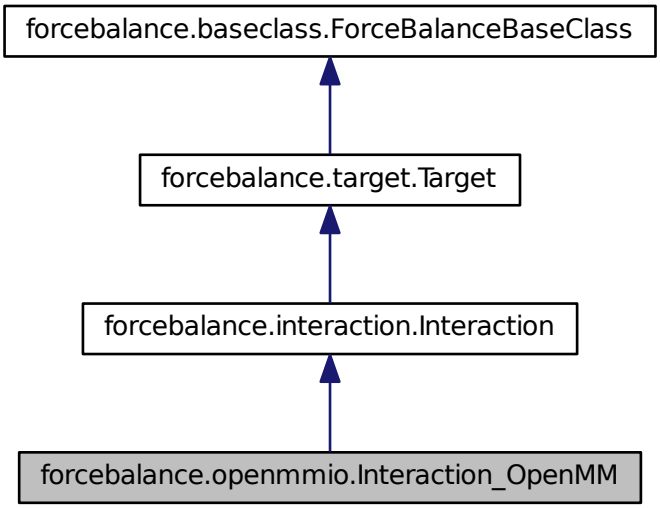
The documentation for this class was generated from the following file:

- gmxio.py

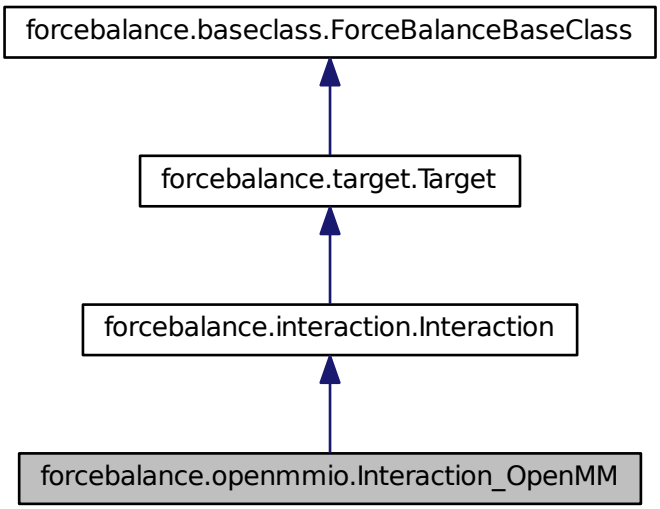
## 8.20 forcebalance.openmmio.Interaction_OpenMM Class Reference

Subclass of Target for interaction matching using OpenMM.

Inheritance diagram for forcebalance.openmmio.Interaction_OpenMM:



Collaboration diagram for forcebalance.openmmio.Interaction_OpenMM:




#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def energy_driver_all
- def interaction_driver_all

#### Public Attributes

- trajfnm

  *Name of the trajectory file containing snapshots.*
- simulations

  *Dictionary of simulation objects (dimer, fraga, fragb)*
- platform

  *Set up three OpenMM System objects.*

### 8.20.1 Detailed Description

Subclass of Target for interaction matching using OpenMM.

Definition at line 534 of file openmmio.py.

### 8.20.2 Constructor & Destructor Documentation

#### 8.20.2.1 def forcebalance.openmmio.Interaction_OpenMM.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 537 of file openmmio.py.

### 8.20.3 Member Function Documentation

#### 8.20.3.1 def forcebalance.openmmio.Interaction_OpenMM.energy_driver_all ( *self, mode* )

Definition at line 598 of file openmmio.py.

Here is the call graph for this function:


#### 8.20.3.2 def forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all ( *self, dielectric* = `False` )

Definition at line 651 of file openmmio.py.

#### 8.20.3.3 def forcebalance.openmmio.Interaction_OpenMM.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 545 of file openmmio.py.

### 8.20.4 Member Data Documentation

#### 8.20.4.1 forcebalance::openmmio.Interaction_OpenMM::platform

Set up three OpenMM System objects.

Set the device to the environment variable or zero otherwise.

TODO: The following code should not be repeated everywhere. Set the simulation platform

Definition at line 548 of file openmmio.py.

#### 8.20.4.2 forcebalance::openmmio.Interaction_OpenMM::simulations

Dictionary of simulation objects (dimer, fraga, fragb)

Definition at line 539 of file openmmio.py.

#### 8.20.4.3 forcebalance::openmmio.Interaction_OpenMM::trajfnm

Name of the trajectory file containing snapshots.

Definition at line 538 of file openmmio.py.

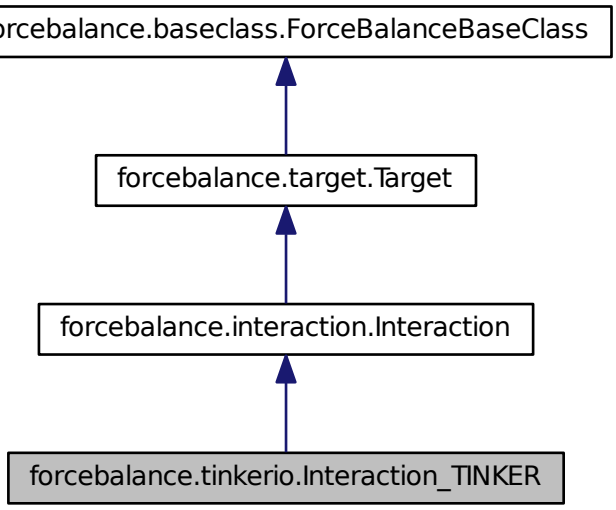
The documentation for this class was generated from the following file:

- openmmio.py

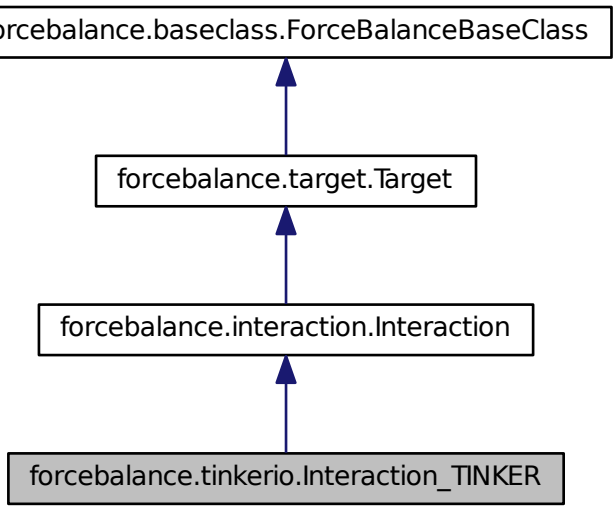
## 8.21 forcebalance.tinkerio.Interaction_TINKER Class Reference

Subclass of Target for interaction matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Interaction_TINKER:



Collaboration diagram for forcebalance.tinkerio.Interaction_TINKER:




#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def prepare_temp_directory

  *Prepare the temporary directory, by default does nothing.*
- def energy_driver_all
- def interaction_driver_all

#### Public Attributes

- trajfnm

  *Name of the trajectory.*

### 8.21.1 Detailed Description

Subclass of Target for interaction matching using TINKER.

Definition at line 582 of file tinkerio.py.

### 8.21.2 Constructor & Destructor Documentation

#### 8.21.2.1 def forcebalance.tinkerio.Interaction_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 585 of file tinkerio.py.

### 8.21.3 Member Function Documentation

#### 8.21.3.1 def forcebalance.tinkerio.Interaction_TINKER.energy_driver_all ( *self, select* = `None` )

Definition at line 598 of file tinkerio.py.

Here is the call graph for this function:

#### 8.21.3.2 def forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all ( *self, dielectric* = `False` )

Definition at line 615 of file tinkerio.py.

#### 8.21.3.3 def forcebalance.tinkerio.Interaction_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Reimplemented from forcebalance.interaction.Interaction.

Definition at line 590 of file tinkerio.py.

### 8.21.4 Member Data Documentation

#### 8.21.4.1 forcebalance::tinkerio.Interaction_TINKER::trajfnm

Name of the trajectory.

Definition at line 586 of file tinkerio.py.

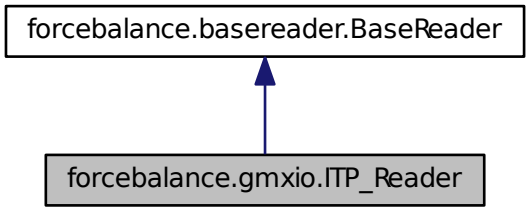
The documentation for this class was generated from the following file:

- tinkerio.py

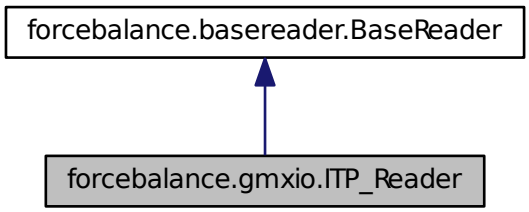
## 8.22 forcebalance.gmxio.ITP_Reader Class Reference

Finite state machine for parsing GROMACS force field files.

Inheritance diagram for forcebalance.gmxio.ITP_Reader:




Collaboration diagram for forcebalance.gmxio.ITP_Reader:




#### Public Member Functions

- def __init__
- def feed

  *Given a line, determine the interaction type and the atoms involved (the suffix).*

#### Public Attributes

- sec

  *The current section that we're in.*
- nbtype

  *Nonbonded type.*
- mol

  *The current molecule (set by the moleculetype keyword)*
- pdict

  *The parameter dictionary (defined in this file)*
- atomnames

*Listing of all atom names in the file, (probably unnecessary)*

- atomtypes

  *Listing of all atom types in the file, (probably unnecessary)*

- atomtype_to_mass

  *A dictionary of atomic masses.*

- itype
- suffix
- molatom

  *The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.*

#### 8.22.1 Detailed Description

Finite state machine for parsing GROMACS force field files.

We open the force field file and read all of its lines. As we loop through the force field file, we look for two types of tags: (1) section markers, in GMX indicated by [ section_name ], which allows us to determine the section, and (2) parameter tags, indicated by the 'PARM' or 'RPT' keywords.

As we go through the file, we figure out the atoms involved in the interaction described on each line.

When a 'PARM' keyword is indicated, it is followed by a number which is the field in the line to be modified, starting with zero. Based on the field number and the section name, we can figure out the parameter type. With the parameter type and the atoms in hand, we construct a 'parameter identifier' or pid which uniquely identifies that parameter. We also store the physical parameter value in an array called 'pvals0' and the precise location of that parameter (by filename, line number, and field number) in a list called 'pfields'.

An example: Suppose in 'my_ff.itp' I encounter the following on lines 146 and 147:

```
[ angletypes ]
CA   CB   O   1   109.47  350.00  ; PARM 4 5
```

From reading `[ angletypes ]` I know I'm in the 'angletypes' section.

On the next line, I notice two parameters on fields 4 and 5.

From the atom types, section type and field number I know the parameter IDs are `'ANGLESBCACBO'` and `'ANGLESKCACBO'`.

After building `map={'ANGLESBCACBO':1,'ANGLESKCACBO':2}`, I store the values in an array: `pvals0=array([109.47,350.00])`, and I put the parameter locations in pfields: `pfields=[['my_ff.itp',147,4,1.0],['my_ff.itp',146,5,1.0]]`. The 1.0 is a 'multiplier' and I will explain it below.

Note that in the creation of parameter IDs, we run into the issue that the atoms involved in the interaction may be labeled in reverse order (e.g. `OCACB`). Thus, we store both the normal and the reversed parameter ID in the map.

Parameter repetition and multiplier:

If `'RPT'` is encountered in the line, it is always in the syntax: `'RPT 4 ANGLESBCACAH 5 MINUS_ANGLESKCACAH /RPT'`. In this case, field 4 is replaced by the stored parameter value corresponding to `ANGLESBCACAH` and field 5 is replaced by -1 times the stored value of `ANGLESKCACAH`. Now I just picked this as an example, I don't think people actually want a negative angle force constant .. :) the `MINUS` keyword does come in handy for assigning atomic charges and virtual site positions. In order to achieve this, a multiplier of -1.0 is stored into pfields instead of 1.0.

**Todo** Note that I can also create the opposite virtual site position by changing the atom labeling, woo!

Definition at line 272 of file gmxio.py.

### 8.22.2 Constructor & Destructor Documentation

#### 8.22.2.1 def forcebalance.gmxio.ITP_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 275 of file gmxio.py.

### 8.22.3 Member Function Documentation

#### 8.22.3.1 def forcebalance.gmxio.ITP_Reader.feed ( *self, line* )

Given a line, determine the interaction type and the atoms involved (the suffix).

For example, we want

```
H O H 5 1.231258497536e+02 4.269161426840e+02 -1.033397697685e-02 1.304674117410e+04
; PARM 4 5 6 7
```

to give us itype = 'UREY_BRADLEY' and suffix = 'HOH'

If we are in a TypeSection, it returns a list of atom types;

If we are in a TopolSection, it returns a list of atom names.

The section is essentially a case statement that picks out the appropriate interaction type and makes a list of the atoms involved

Note that we can call gmxdump for this as well, but I prefer to read the force field file directly.

ToDo: [ atoms ] section might need to be more flexible to accommodate optional fields

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 313 of file gmxio.py.

### 8.22.4 Member Data Documentation

#### 8.22.4.1 forcebalance::gmxio.ITP_Reader::atomnames

Listing of all atom names in the file, (probably unnecessary)

Definition at line 280 of file gmxio.py.

#### 8.22.4.2 forcebalance::gmxio.ITP_Reader::atomtype_to_mass

A dictionary of atomic masses.

Definition at line 282 of file gmxio.py.

#### 8.22.4.3 forcebalance::gmxio.ITP_Reader::atomtypes

Listing of all atom types in the file, (probably unnecessary)

Definition at line 281 of file gmxio.py.

#### 8.22.4.4 forcebalance::gmxio.ITP_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 313 of file gmxio.py.

#### 8.22.4.5 forcebalance::gmxio.ITP_Reader::mol

The current molecule (set by the moleculetype keyword)

Definition at line 278 of file gmxio.py.

#### 8.22.4.6 forcebalance::gmxio.ITP_Reader::molatom

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 313 of file gmxio.py.

#### 8.22.4.7 forcebalance::gmxio.ITP_Reader::nbtype

Nonbonded type.

Definition at line 277 of file gmxio.py.

#### 8.22.4.8 forcebalance::gmxio.ITP_Reader::pdict

The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 279 of file gmxio.py.

#### 8.22.4.9 forcebalance::gmxio.ITP_Reader::sec

The current section that we're in.

Definition at line 276 of file gmxio.py.

#### 8.22.4.10 forcebalance::gmxio.ITP_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 313 of file gmxio.py.

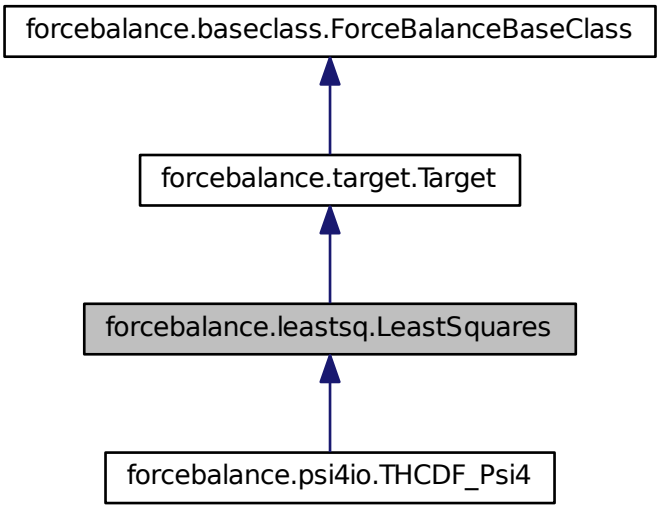
The documentation for this class was generated from the following file:

- gmxio.py

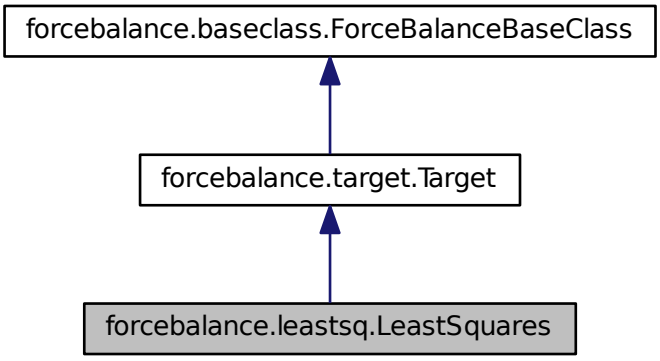
## 8.23 forcebalance.leastsq.LeastSquares Class Reference

Subclass of Target for general least squares fitting.

Inheritance diagram for forcebalance.leastsq.LeastSquares:




Collaboration diagram for forcebalance.leastsq.LeastSquares:




**Public Member Functions**

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*

- def indicate
- def get

  *LPW 05-30-2012.*

**Public Attributes**

- call_derivatives

  *Number of snapshots.*
- MAQ

  *Dictionary for derivative terms.*
- D
- objective

### 8.23.1 Detailed Description

Subclass of Target for general least squares fitting.

Definition at line 32 of file leastsq.py.

### 8.23.2 Constructor & Destructor Documentation

#### 8.23.2.1 def forcebalance.leastsq.LeastSquares.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.psi4io.THCDF_Psi4.

Definition at line 35 of file leastsq.py.

### 8.23.3 Member Function Documentation

#### 8.23.3.1 def forcebalance.leastsq.LeastSquares.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

LPW 05-30-2012.

This subroutine builds the objective function (and optionally its derivatives) from a general software.

This subroutine interfaces with simulation software 'drivers'. The driver is expected to give exact values, fitting values, and weights.

**Parameters**

| | | |
|---|---|---|
| `in` | *mvals* | Mathematical parameter values |
| `in` | *AGrad* | Switch to turn on analytic gradient |
| `in` | *AHess* | Switch to turn on analytic Hessian |

**Returns**

Answer Contribution to the objective function

Reimplemented from forcebalance.target.Target.

Definition at line 72 of file leastsq.py.

#### 8.23.3.2 def forcebalance.leastsq.LeastSquares.indicate ( *self* )

Reimplemented in forcebalance.psi4io.THCDF_Psi4.

Definition at line 50 of file leastsq.py.

Here is the call graph for this function:


### 8.23.4 Member Data Documentation

#### 8.23.4.1 forcebalance::leastsq.LeastSquares::call_derivatives

Number of snapshots.

Which parameters are differentiated?

Definition at line 37 of file leastsq.py.

#### 8.23.4.2 forcebalance::leastsq.LeastSquares::D

Definition at line 73 of file leastsq.py.

#### 8.23.4.3 forcebalance::leastsq.LeastSquares::MAQ

Dictionary for derivative terms.

Definition at line 73 of file leastsq.py.

#### 8.23.4.4 forcebalance::leastsq.LeastSquares::objective

Definition at line 73 of file leastsq.py.

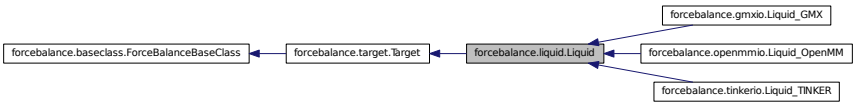
The documentation for this class was generated from the following file:

- leastsq.py

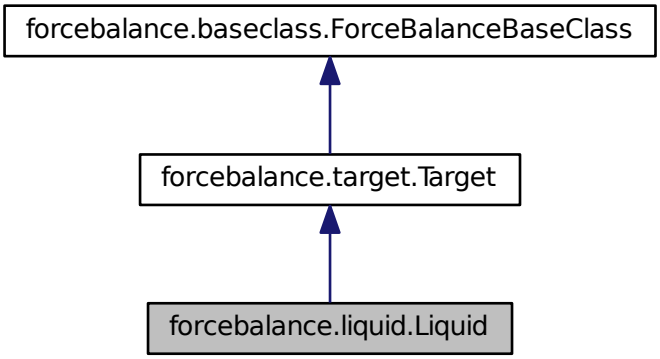
## 8.24 forcebalance.liquid.Liquid Class Reference

Subclass of Target for liquid property matching.

Inheritance diagram for forcebalance.liquid.Liquid:




Collaboration diagram for forcebalance.liquid.Liquid:




### Public Member Functions

- def __init__

  *Instantiation of the subclass.*
- def read_data
- def npt_simulation

  *Submit a NPT simulation to the Work Queue.*
- def indicate
- def objective_term
- def submit_jobs
- def get

  *Fitting of liquid bulk properties.*
- def polarization_correction

  *Return the self-polarization correction as described in Berendsen et al., JPC 1987.*

### Public Attributes

- do_self_pol
- SavedMVal

*Read the reference data.*

- SavedTraj

  *Saved trajectories for all iterations and all temperatures :)*

- MBarEnergy

  *Evaluated energies for all trajectories (i.e.*

- nptpfx
- nptfiles
- nptsfx
- RefData
- PhasePoints
- Labels
- w_rho

  *Density.*

- w_hvap
- w_alpha
- w_kappa
- w_cp
- w_eps0

#### 8.24.1 Detailed Description

Subclass of Target for liquid property matching.

Definition at line 48 of file liquid.py.

#### 8.24.2 Constructor & Destructor Documentation

##### 8.24.2.1 def forcebalance.liquid.Liquid.__init__ ( *self, options, tgt_opts, forcefield* )

Instantiation of the subclass.

We begin by instantiating the superclass here and also defining a number of core concepts for energy / force matching.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.gmxio.Liquid_GMX, forcebalance.openmmio.Liquid_OpenMM, and forcebalance.tinkerio.Liquid_TINKER.

Definition at line 61 of file liquid.py.

#### 8.24.3 Member Function Documentation

##### 8.24.3.1 def forcebalance.liquid.Liquid.get ( *self, mvals, AGrad* = `True`, *AHess* = `True` )

Fitting of liquid bulk properties.

This is the current major direction of development for ForceBalance. Basically, fitting the QM energies / forces alone does not always give us the best simulation behavior. In many cases it makes more sense to try and reproduce some experimentally known data as well.

In order to reproduce experimentally known data, we need to run a simulation and compare the simulation result to experiment. The main challenge here is that the simulations are computationally intensive (i.e. they require energy and force evaluations), and furthermore the results are noisy. We need to run the simulations automatically and remotely (i.e. on clusters) and a good way to calculate the derivatives of the simulation results with respect to the parameter values.

This function contains some experimentally known values of the density and enthalpy of vaporization (Hvap) of liquid water. It launches the density and Hvap calculations on the cluster, and gathers the results / derivatives. The actual calculation of results / derivatives is done in a separate file.

After the results come back, they are gathered together to form an objective function.

**Parameters**

| | | |
|---|---|---|
| in | *mvals* | Mathematical parameter values |
| in | *AGrad* | Switch to turn on analytic gradient |
| in | *AHess* | Switch to turn on analytic Hessian |

**Returns**

Answer Contribution to the objective function

Reimplemented from forcebalance.target.Target.

Definition at line 392 of file liquid.py.

#### 8.24.3.2 def forcebalance.liquid.Liquid.indicate ( *self* )

Definition at line 252 of file liquid.py.

#### 8.24.3.3 def forcebalance.liquid.Liquid.npt_simulation ( *self, temperature, pressure, simnum* )

Submit a NPT simulation to the Work Queue.

Reimplemented in forcebalance.tinkerio.Liquid_TINKER.

Definition at line 232 of file liquid.py.

Here is the call graph for this function:


#### 8.24.3.4 def forcebalance.liquid.Liquid.objective_term ( *self, points, expname, calc, err, grad, name =* `"Quantity"`*, SubAverage =* `False` )

Definition at line 256 of file liquid.py.

Here is the call graph for this function:

#### 8.24.3.5 def forcebalance.liquid.Liquid.polarization_correction ( *self, mvals* )

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

Reimplemented in forcebalance.gmxio.Liquid_GMX, and forcebalance.openmmio.Liquid_OpenMM.

Definition at line 724 of file liquid.py.

#### 8.24.3.6 def forcebalance.liquid.Liquid.read_data ( *self* )

Definition at line 140 of file liquid.py.

#### 8.24.3.7 def forcebalance.liquid.Liquid.submit_jobs ( *self, mvals, AGrad* = `True`, *AHess* = `True` )

Reimplemented from forcebalance.target.Target.

Definition at line 334 of file liquid.py.

Here is the call graph for this function:


### 8.24.4 Member Data Documentation

#### 8.24.4.1 forcebalance::liquid.Liquid::do_self_pol

Definition at line 61 of file liquid.py.

#### 8.24.4.2 forcebalance::liquid.Liquid::Labels

Definition at line 140 of file liquid.py.

#### 8.24.4.3 forcebalance::liquid.Liquid::MBarEnergy

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 66 of file liquid.py.

#### 8.24.4.4 forcebalance::liquid.Liquid::nptfiles

Definition at line 66 of file liquid.py.

#### 8.24.4.5 forcebalance::liquid.Liquid::nptpfx

Reimplemented in forcebalance.gmxio.Liquid_GMX.

Definition at line 66 of file liquid.py.

#### 8.24.4.6 forcebalance::liquid.Liquid::nptsfx

Definition at line 66 of file liquid.py.

#### 8.24.4.7 forcebalance::liquid.Liquid::PhasePoints

Definition at line 140 of file liquid.py.

#### 8.24.4.8 forcebalance::liquid.Liquid::RefData

Definition at line 140 of file liquid.py.

#### 8.24.4.9 forcebalance::liquid.Liquid::SavedMVal

Read the reference data.

Prepare the temporary directory Saved force field mvals for all iterations

Definition at line 64 of file liquid.py.

#### 8.24.4.10 forcebalance::liquid.Liquid::SavedTraj

Saved trajectories for all iterations and all temperatures :)

Definition at line 65 of file liquid.py.

#### 8.24.4.11 forcebalance::liquid.Liquid::w_alpha

Definition at line 399 of file liquid.py.

#### 8.24.4.12 forcebalance::liquid.Liquid::w_cp

Definition at line 399 of file liquid.py.

#### 8.24.4.13 forcebalance::liquid.Liquid::w_eps0

Definition at line 399 of file liquid.py.

#### 8.24.4.14 forcebalance::liquid.Liquid::w_hvap

Definition at line 399 of file liquid.py.

#### 8.24.4.15 forcebalance::liquid.Liquid::w_kappa

Definition at line 399 of file liquid.py.

#### 8.24.4.16 forcebalance::liquid.Liquid::w_rho

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

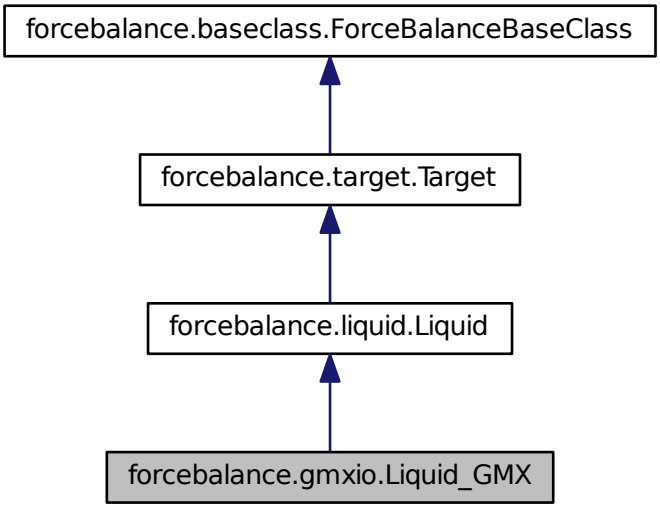
Definition at line 399 of file liquid.py.

The documentation for this class was generated from the following file:

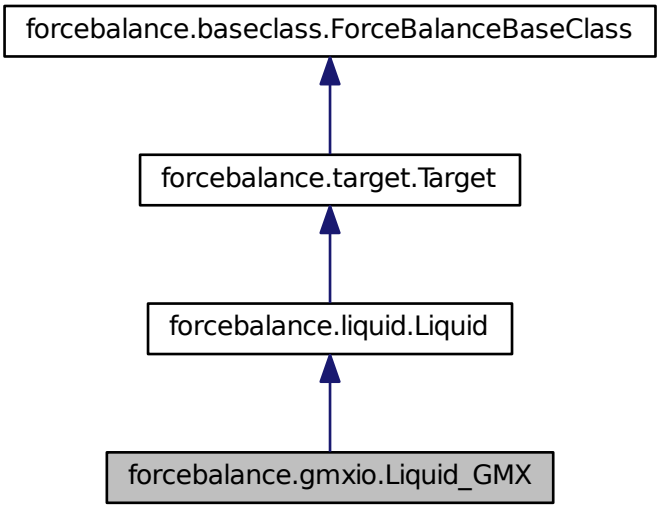
- liquid.py

## 8.25 forcebalance.gmxio.Liquid_GMX Class Reference

Inheritance diagram for forcebalance.gmxio.Liquid_GMX:




Collaboration diagram for forcebalance.gmxio.Liquid_GMX:



### Public Member Functions

- def __init__
  *Instantiation of the subclass.*
- def prepare_temp_directory
  *Prepare the temporary directory by copying in important files.*
- def polarization_correction
  *Return the self-polarization correction as described in Berendsen et al., JPC 1987.*

### Public Attributes

- liquid_fnm
- liquid_conf
- liquid_traj
- gas_fnm
- nptpfx
- engine

#### 8.25.1 Detailed Description

Definition at line 570 of file gmxio.py.

#### 8.25.2 Constructor & Destructor Documentation

##### 8.25.2.1 def forcebalance.gmxio.Liquid_GMX.__init__ ( *self, options, tgt_opts, forcefield* )

Instantiation of the subclass.

We begin by instantiating the superclass here and also defining a number of core concepts for energy / force matching.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 571 of file gmxio.py.

#### 8.25.3 Member Function Documentation

##### 8.25.3.1 def forcebalance.gmxio.Liquid_GMX.polarization_correction ( *self, mvals* )

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 612 of file gmxio.py.

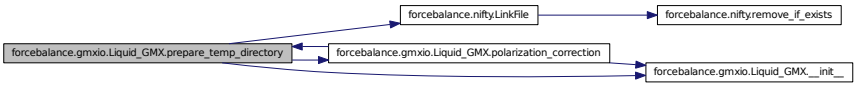
Here is the call graph for this function:

#### 8.25.3.2 def forcebalance.gmxio.Liquid_GMX.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory by copying in important files.

Definition at line 595 of file gmxio.py.

Here is the call graph for this function:




### 8.25.4 Member Data Documentation

#### 8.25.4.1 forcebalance::gmxio.Liquid_GMX::engine

Definition at line 571 of file gmxio.py.

#### 8.25.4.2 forcebalance::gmxio.Liquid_GMX::gas_fnm

Definition at line 571 of file gmxio.py.

#### 8.25.4.3 forcebalance::gmxio.Liquid_GMX::liquid_conf

Definition at line 571 of file gmxio.py.

#### 8.25.4.4 forcebalance::gmxio.Liquid_GMX::liquid_fnm

Definition at line 571 of file gmxio.py.

#### 8.25.4.5 forcebalance::gmxio.Liquid_GMX::liquid_traj

Definition at line 571 of file gmxio.py.

#### 8.25.4.6 forcebalance::gmxio.Liquid_GMX::nptpfx

Reimplemented from forcebalance.liquid.Liquid.

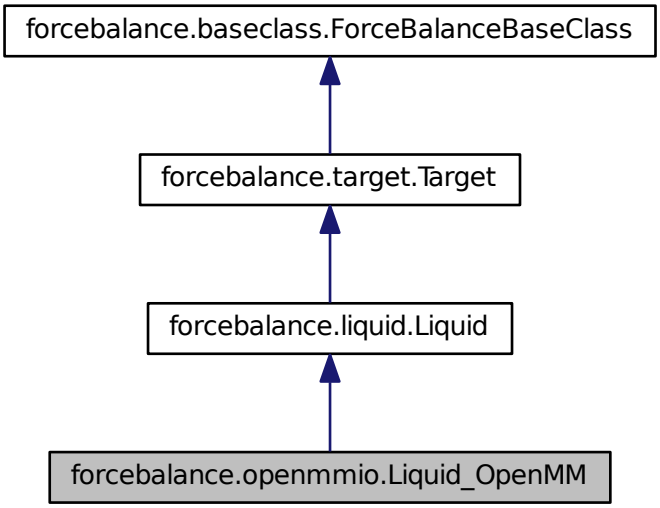
Definition at line 571 of file gmxio.py.

The documentation for this class was generated from the following file:

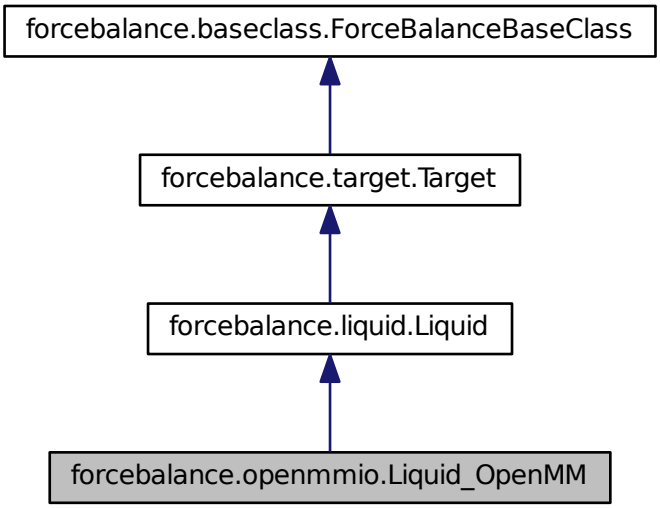
- gmxio.py

## 8.26 forcebalance.openmmio.Liquid_OpenMM Class Reference

Inheritance diagram for forcebalance.openmmio.Liquid_OpenMM:




Collaboration diagram for forcebalance.openmmio.Liquid_OpenMM:



#### Public Member Functions

- def __init__
  *Instantiation of the subclass.*
- def prepare_temp_directory
  *Prepare the temporary directory by copying in important files.*
- def polarization_correction
  *Return the self-polarization correction as described in Berendsen et al., JPC 1987.*

#### Public Attributes

- mpdb
- platform
- msim
- liquid_fnm
- liquid_conf
- liquid_traj
- gas_fnm
- engine

### 8.26.1 Detailed Description

Definition at line 327 of file openmmio.py.

### 8.26.2 Constructor & Destructor Documentation

#### 8.26.2.1 def forcebalance.openmmio.Liquid_OpenMM.__init__ ( *self, options, tgt_opts, forcefield* )

Instantiation of the subclass.

We begin by instantiating the superclass here and also defining a number of core concepts for energy / force matching.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 328 of file openmmio.py.

### 8.26.3 Member Function Documentation

#### 8.26.3.1 def forcebalance.openmmio.Liquid_OpenMM.polarization_correction ( *self, mvals* )

Return the self-polarization correction as described in Berendsen et al., JPC 1987.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 375 of file openmmio.py.

Here is the call graph for this function:

#### 8.26.3.2 def forcebalance.openmmio.Liquid_OpenMM.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory by copying in important files.

Definition at line 368 of file openmmio.py.

Here is the call graph for this function:


### 8.26.4 Member Data Documentation

#### 8.26.4.1 forcebalance::openmmio.Liquid_OpenMM::engine

Definition at line 328 of file openmmio.py.

#### 8.26.4.2 forcebalance::openmmio.Liquid_OpenMM::gas_fnm

Definition at line 328 of file openmmio.py.

#### 8.26.4.3 forcebalance::openmmio.Liquid_OpenMM::liquid_conf

Definition at line 328 of file openmmio.py.

#### 8.26.4.4 forcebalance::openmmio.Liquid_OpenMM::liquid_fnm

Definition at line 328 of file openmmio.py.

#### 8.26.4.5 forcebalance::openmmio.Liquid_OpenMM::liquid_traj

Definition at line 328 of file openmmio.py.

#### 8.26.4.6 forcebalance::openmmio.Liquid_OpenMM::mpdb

Definition at line 328 of file openmmio.py.

#### 8.26.4.7 forcebalance::openmmio.Liquid_OpenMM::msim

Definition at line 328 of file openmmio.py.

#### 8.26.4.8 forcebalance::openmmio.Liquid_OpenMM::platform

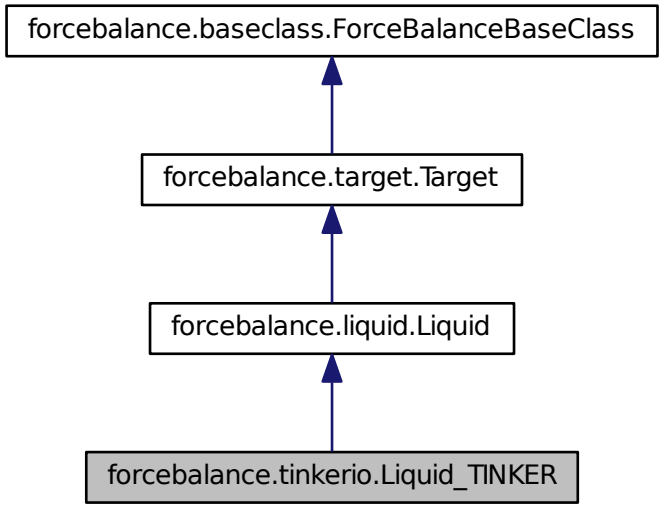
Definition at line 328 of file openmmio.py.

The documentation for this class was generated from the following file:

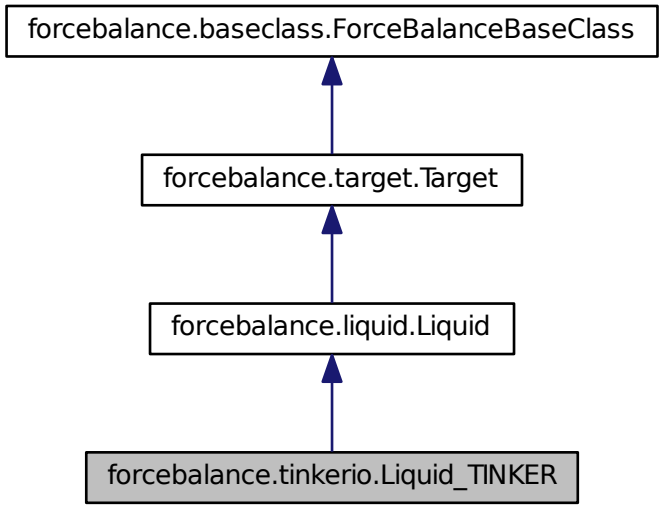
- openmmio.py

## 8.27 forcebalance.tinkerio.Liquid_TINKER Class Reference

Inheritance diagram for forcebalance.tinkerio.Liquid_TINKER:




Collaboration diagram for forcebalance.tinkerio.Liquid_TINKER:



#### Public Member Functions

- def __init__
  *Instantiation of the subclass.*
- def prepare_temp_directory
  *Prepare the temporary directory by copying in important files.*
- def npt_simulation
  *Submit a NPT simulation to the Work Queue.*

#### Public Attributes

- DynDict
- DynDict_New

### 8.27.1 Detailed Description

Definition at line 232 of file tinkerio.py.

### 8.27.2 Constructor & Destructor Documentation

#### 8.27.2.1 def forcebalance.tinkerio.Liquid_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

Instantiation of the subclass.

We begin by instantiating the superclass here and also defining a number of core concepts for energy / force matching.

**Todo** Obtain the number of true atoms (or the particle -> atom mapping) from the force field.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 233 of file tinkerio.py.

### 8.27.3 Member Function Documentation

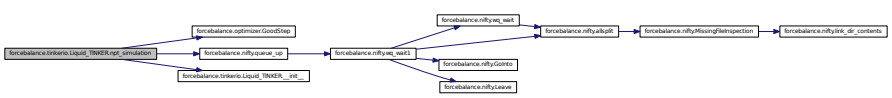
#### 8.27.3.1 def forcebalance.tinkerio.Liquid_TINKER.npt_simulation ( *self, temperature, pressure, simnum* )

Submit a NPT simulation to the Work Queue.

Reimplemented from forcebalance.liquid.Liquid.

Definition at line 260 of file tinkerio.py.

Here is the call graph for this function:

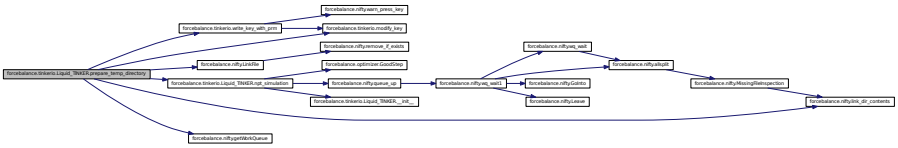


#### 8.27.3.2 def forcebalance.tinkerio.Liquid_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory by copying in important files.

Definition at line 242 of file tinkerio.py.

Here is the call graph for this function:



### 8.27.4 Member Data Documentation

#### 8.27.4.1 forcebalance::tinkerio.Liquid_TINKER::DynDict

Definition at line 233 of file tinkerio.py.

#### 8.27.4.2 forcebalance::tinkerio.Liquid_TINKER::DynDict_New

Definition at line 233 of file tinkerio.py.

The documentation for this class was generated from the following file:

- tinkerio.py

## 8.28 forcebalance.Mol2.mol2 Class Reference

This is to manage one mol2 series of lines on the form:

### Public Member Functions

- def __init__
- def __repr__
- def out
- def set_mol_name

  *bond identifier (integer, starting from 1)*
- def set_num_atoms

  *number of atoms (integer)*
- def set_num_bonds

  *number of bonds (integer)*
- def set_num_subst

  *number of substructures (integer)*
- def set_num_feat

  *number of features (integer)*
- def set_num_sets

  *number of sets (integer)*
- def set_mol_type

*bond identifier (integer, starting from 1)*

- def set_charge_type

  *bond identifier (integer, starting from 1)*

- def parse

  *Parse a series of text lines, and setup compound information.*

- def get_atom

  *return the atom instance given its atom identifier*

- def get_bonded_atoms

  *return a dictionnary of atom instances bonded to the atom, and their types*

- def set_donnor_acceptor_atoms

  *modify atom types to specify donnor, acceptor, or both*

#### Public Attributes

- mol_name
- num_atoms
- num_bonds
- num_subst
- num_feat
- num_sets
- mol_type
- charge_type
- comments
- atoms
- bonds

### 8.28.1 Detailed Description

This is to manage one mol2 series of lines on the form:

```
@<TRIPOS>MOLECULE
CDK2.xray.inh1.1E9H
 34 37 0 0 0
SMALL
GASTEIGER
Energy = 0

@<TRIPOS>ATOM
      1 C1          5.4790   42.2880   49.5910 C.ar    1  <1>         0.0424
      2 C2          4.4740   42.6430   50.5070 C.ar    1  <1>         0.0447
@<TRIPOS>BOND
     1     1     2   ar
     2     1     6   ar
```

Definition at line 288 of file Mol2.py.

### 8.28.2 Constructor & Destructor Documentation

#### 8.28.2.1 def forcebalance.Mol2.mol2.__init__ ( *self, data* )

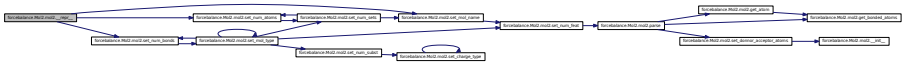
Definition at line 289 of file Mol2.py.

### 8.28.3 Member Function Documentation

#### 8.28.3.1 def forcebalance.Mol2.mol2.__repr__ ( *self* )

Definition at line 305 of file Mol2.py.

Here is the call graph for this function:



#### 8.28.3.2 def forcebalance.Mol2.mol2.get_atom ( *self, id* )

return the atom instance given its atom identifier

Definition at line 461 of file Mol2.py.

Here is the call graph for this function:


#### 8.28.3.3 def forcebalance.Mol2.mol2.get_bonded_atoms ( *self, id* )

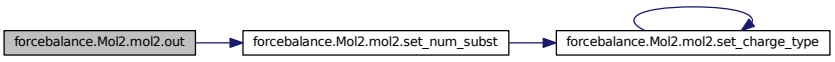
return a dictionnary of atom instances bonded to the atom, and their types

Definition at line 475 of file Mol2.py.

#### 8.28.3.4 def forcebalance.Mol2.mol2.out ( *self, f* = `sys.stdout` )

Definition at line 325 of file Mol2.py.

Here is the call graph for this function:

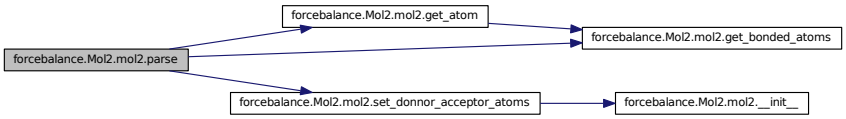



#### 8.28.3.5 def forcebalance.Mol2.mol2.parse ( *self, data* )

Parse a series of text lines, and setup compound information.

Definition at line 405 of file Mol2.py.

Here is the call graph for this function:




#### 8.28.3.6 def forcebalance.Mol2.mol2.set_charge_type ( *self, charge_type* = None )

bond identifier (integer, starting from 1)

Definition at line 395 of file Mol2.py.

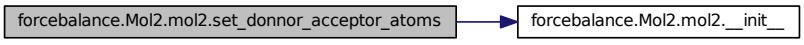
Here is the call graph for this function:


#### 8.28.3.7 def forcebalance.Mol2.mol2.set_donnor_acceptor_atoms ( *self, verbose* = 0 )

modify atom types to specify donnor, acceptor, or both

Definition at line 490 of file Mol2.py.

Here is the call graph for this function:

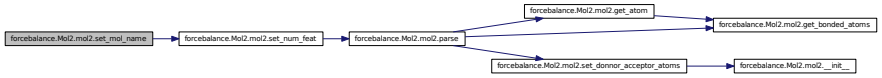



#### 8.28.3.8 def forcebalance.Mol2.mol2.set_mol_name ( *self, mol_name* = None )

bond identifier (integer, starting from 1)

Definition at line 332 of file Mol2.py.

Here is the call graph for this function:




#### 8.28.3.9 def forcebalance.Mol2.mol2.set_mol_type ( *self*, *mol_type* = None )

bond identifier (integer, starting from 1)

Definition at line 386 of file Mol2.py.

Here is the call graph for this function:


#### 8.28.3.10 def forcebalance.Mol2.mol2.set_num_atoms ( *self*, *num_atoms* = None )

number of atoms (integer)

Definition at line 341 of file Mol2.py.

Here is the call graph for this function:


#### 8.28.3.11 def forcebalance.Mol2.mol2.set_num_bonds ( *self*, *num_bonds* = None )

number of bonds (integer)

Definition at line 350 of file Mol2.py.

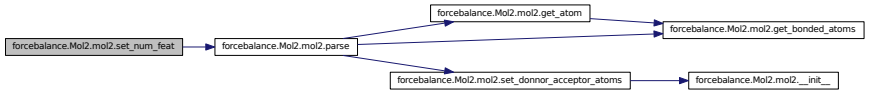
Here is the call graph for this function:


#### 8.28.3.12 def forcebalance.Mol2.mol2.set_num_feat ( *self*, *num_feat* = None )

number of features (integer)

Definition at line 368 of file Mol2.py.

Here is the call graph for this function:




#### 8.28.3.13 def forcebalance.Mol2.mol2.set_num_sets ( *self, num_sets* = None )

number of sets (integer)

Definition at line 377 of file Mol2.py.

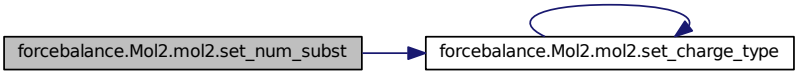
Here is the call graph for this function:


#### 8.28.3.14 def forcebalance.Mol2.mol2.set_num_subst ( *self, num_subst* = None )

number of substructures (integer)

Definition at line 359 of file Mol2.py.

Here is the call graph for this function:




### 8.28.4 Member Data Documentation

#### 8.28.4.1 forcebalance::Mol2.mol2::atoms

Definition at line 289 of file Mol2.py.

#### 8.28.4.2 forcebalance::Mol2.mol2::bonds

Definition at line 289 of file Mol2.py.

#### 8.28.4.3 forcebalance::Mol2.mol2::charge_type

Definition at line 289 of file Mol2.py.

#### 8.28.4.4 forcebalance::Mol2.mol2::comments

Definition at line 289 of file Mol2.py.

#### 8.28.4.5 forcebalance::Mol2.mol2::mol_name

Definition at line 289 of file Mol2.py.

#### 8.28.4.6 forcebalance::Mol2.mol2::mol_type

Definition at line 289 of file Mol2.py.

#### 8.28.4.7 forcebalance::Mol2.mol2::num_atoms

Definition at line 289 of file Mol2.py.

#### 8.28.4.8 forcebalance::Mol2.mol2::num_bonds

Definition at line 289 of file Mol2.py.

#### 8.28.4.9 forcebalance::Mol2.mol2::num_feat

Definition at line 289 of file Mol2.py.

#### 8.28.4.10 forcebalance::Mol2.mol2::num_sets

Definition at line 289 of file Mol2.py.

#### 8.28.4.11 forcebalance::Mol2.mol2::num_subst

Definition at line 289 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

## 8.29 forcebalance.Mol2.mol2_atom Class Reference

This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.

### Public Member Functions

- def __init__
  *if data is passed, it will be installed*
- def parse
  *split the text line into a series of properties*
- def __repr__
  *assemble the properties as a text line, and return it*
- def set_atom_id
  *atom identifier (integer, starting from 1)*
- def set_atom_name
  *The name of the atom (string)*
- def set_crds
  *the coordinates of the atom*
- def set_atom_type
  *The mol2 type of the atom.*

- def set_subst_id

  *substructure identifier*
- def set_subst_name

  *substructure name*
- def set_charge

  *atomic charge*
- def set_status_bit

  *Never to use (in theory)*

#### Public Attributes

- atom_id
- atom_name
- x
- y
- z
- atom_type
- subst_id
- subst_name
- charge
- status_bit

### 8.29.1 Detailed Description

This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.

Definition at line 32 of file Mol2.py.

### 8.29.2 Constructor & Destructor Documentation

#### 8.29.2.1 def forcebalance.Mol2.mol2_atom.__init__ ( *self*, *data* = None )

if data is passed, it will be installed

Definition at line 37 of file Mol2.py.

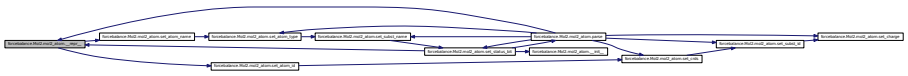
### 8.29.3 Member Function Documentation

#### 8.29.3.1 def forcebalance.Mol2.mol2_atom.__repr__ ( *self* )

assemble the properties as a text line, and return it

Definition at line 78 of file Mol2.py.

Here is the call graph for this function:

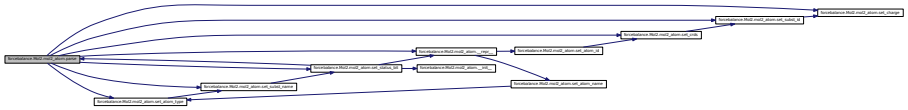


#### 8.29.3.2 def forcebalance.Mol2.mol2_atom.parse ( *self, data* )

split the text line into a series of properties

Definition at line 56 of file Mol2.py.

Here is the call graph for this function:



#### 8.29.3.3 def forcebalance.Mol2.mol2_atom.set_atom_id ( *self, atom_id* = None )

atom identifier (integer, starting from 1)

Definition at line 90 of file Mol2.py.

Here is the call graph for this function:


#### 8.29.3.4 def forcebalance.Mol2.mol2_atom.set_atom_name ( *self, atom_name* = None )

The name of the atom (string)

Definition at line 99 of file Mol2.py.

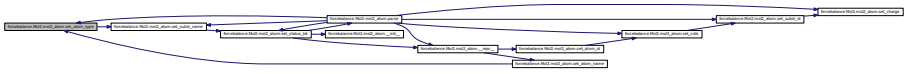
Here is the call graph for this function:

#### 8.29.3.5 def forcebalance.Mol2.mol2_atom.set_atom_type ( *self, atom_type* = None )

The mol2 type of the atom.

Definition at line 119 of file Mol2.py.

Here is the call graph for this function:



#### 8.29.3.6 def forcebalance.Mol2.mol2_atom.set_charge ( *self, charge* = None )

atomic charge

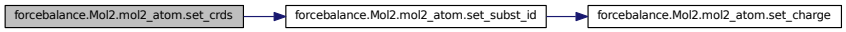
Definition at line 146 of file Mol2.py.

#### 8.29.3.7 def forcebalance.Mol2.mol2_atom.set_crds ( *self*, *x* = None, *y* = None, *z* = None )

the coordinates of the atom

Definition at line 108 of file Mol2.py.

Here is the call graph for this function:

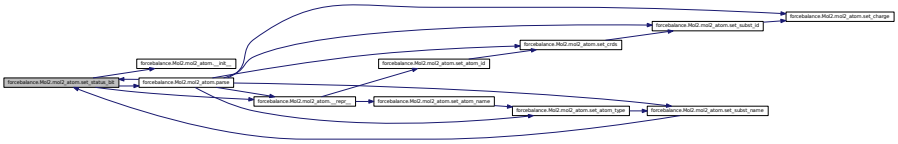



#### 8.29.3.8 def forcebalance.Mol2.mol2_atom.set_status_bit ( *self*, *status_bit* = None )

Never to use (in theory)

Definition at line 155 of file Mol2.py.

Here is the call graph for this function:



#### 8.29.3.9 def forcebalance.Mol2.mol2_atom.set_subst_id ( *self*, *subst_id* = None )

substructure identifier

Definition at line 128 of file Mol2.py.

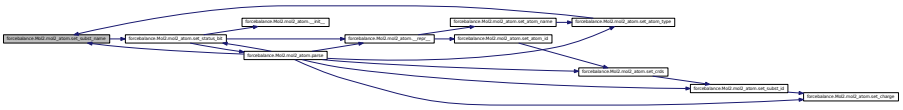
Here is the call graph for this function:


#### 8.29.3.10 def forcebalance.Mol2.mol2_atom.set_subst_name ( *self*, *subst_name* = None )

substructure name

Definition at line 137 of file Mol2.py.

Here is the call graph for this function:



#### 8.29.4 Member Data Documentation

##### 8.29.4.1 forcebalance::Mol2.mol2_atom::atom_id

Definition at line 37 of file Mol2.py.

##### 8.29.4.2 forcebalance::Mol2.mol2_atom::atom_name

Definition at line 37 of file Mol2.py.

##### 8.29.4.3 forcebalance::Mol2.mol2_atom::atom_type

Definition at line 37 of file Mol2.py.

##### 8.29.4.4 forcebalance::Mol2.mol2_atom::charge

Definition at line 37 of file Mol2.py.

##### 8.29.4.5 forcebalance::Mol2.mol2_atom::status_bit

Definition at line 37 of file Mol2.py.

##### 8.29.4.6 forcebalance::Mol2.mol2_atom::subst_id

Definition at line 37 of file Mol2.py.

##### 8.29.4.7 forcebalance::Mol2.mol2_atom::subst_name

Definition at line 37 of file Mol2.py.

##### 8.29.4.8 forcebalance::Mol2.mol2_atom::x

Definition at line 37 of file Mol2.py.

##### 8.29.4.9 forcebalance::Mol2.mol2_atom::y

Definition at line 37 of file Mol2.py.

##### 8.29.4.10 forcebalance::Mol2.mol2_atom::z

Definition at line 37 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

## 8.30 forcebalance.Mol2.mol2_bond Class Reference

This is to manage mol2 bond lines on the form: 1 1 2 ar.

### Public Member Functions

- def __init__
  *if data is passed, it will be installed*
- def __repr__
- def parse
  *split the text line into a series of properties*
- def set_bond_id
  *bond identifier (integer, starting from 1)*
- def set_origin_atom_id
  *the origin atom identifier (integer)*
- def set_target_atom_id
  *the target atom identifier (integer)*
- def set_bond_type
  *bond type (string) one of: 1 = single 2 = double 3 = triple am = amide ar = aromatic du = dummy un = unknown nc = not connected*
- def set_status_bit
  *Never to use (in theory)*

### Public Attributes

- bond_id
- origin_atom_id
- target_atom_id
- bond_type
- status_bit

### 8.30.1 Detailed Description

This is to manage mol2 bond lines on the form: 1 1 2 ar.

Definition at line 172 of file Mol2.py.

### 8.30.2 Constructor & Destructor Documentation

#### 8.30.2.1 def forcebalance.Mol2.mol2_bond.__init__ ( *self*, *data* = None )

if data is passed, it will be installed

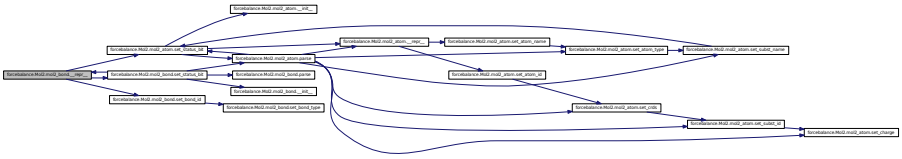
Definition at line 177 of file Mol2.py.

### 8.30.3 Member Function Documentation

#### 8.30.3.1 def forcebalance.Mol2.mol2_bond.__repr__ ( *self* )

Definition at line 186 of file Mol2.py.

Here is the call graph for this function:



#### 8.30.3.2 def forcebalance.Mol2.mol2_bond.parse ( *self, data* )

split the text line into a series of properties

Definition at line 197 of file Mol2.py.

#### 8.30.3.3 def forcebalance.Mol2.mol2_bond.set_bond_id ( *self, bond_id* = None )

bond identifier (integer, starting from 1)

Definition at line 214 of file Mol2.py.

Here is the call graph for this function:


#### 8.30.3.4 def forcebalance.Mol2.mol2_bond.set_bond_type ( *self, bond_type* = None )

bond type (string) one of: 1 = single 2 = double 3 = triple am = amide ar = aromatic du = dummy un = unknown nc = not connected

Definition at line 250 of file Mol2.py.

#### 8.30.3.5 def forcebalance.Mol2.mol2_bond.set_origin_atom_id ( *self, origin_atom_id* = None )

the origin atom identifier (integer)

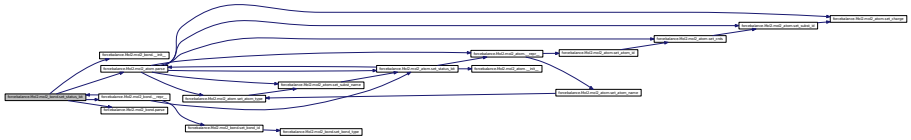
Definition at line 223 of file Mol2.py.

#### 8.30.3.6 def forcebalance.Mol2.mol2_bond.set_status_bit ( *self, status_bit* = None )

Never to use (in theory)

Definition at line 259 of file Mol2.py.

Here is the call graph for this function:

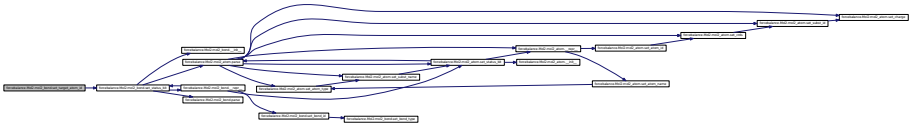


#### 8.30.3.7 def forcebalance.Mol2.mol2_bond.set_target_atom_id ( *self*, *target_atom_id* = None )

the target atom identifier (integer)

Definition at line 232 of file Mol2.py.

Here is the call graph for this function:



### 8.30.4 Member Data Documentation

#### 8.30.4.1 forcebalance::Mol2.mol2_bond::bond_id

Definition at line 177 of file Mol2.py.

#### 8.30.4.2 forcebalance::Mol2.mol2_bond::bond_type

Definition at line 177 of file Mol2.py.

#### 8.30.4.3 forcebalance::Mol2.mol2_bond::origin_atom_id

Definition at line 177 of file Mol2.py.

#### 8.30.4.4 forcebalance::Mol2.mol2_bond::status_bit

Definition at line 197 of file Mol2.py.

#### 8.30.4.5 forcebalance::Mol2.mol2_bond::target_atom_id

Definition at line 177 of file Mol2.py.

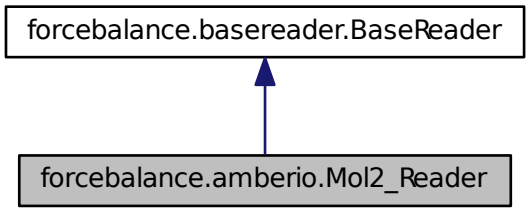
The documentation for this class was generated from the following file:

- Mol2.py

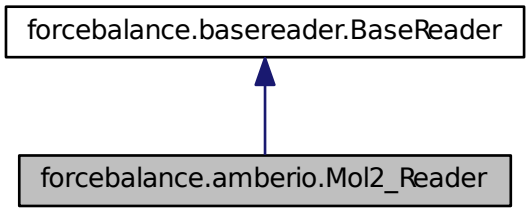
## 8.31 forcebalance.amberio.Mol2_Reader Class Reference

Finite state machine for parsing Mol2 force field file.

Inheritance diagram for forcebalance.amberio.Mol2_Reader:




Collaboration diagram for forcebalance.amberio.Mol2_Reader:




#### Public Member Functions

- def __init__
- def feed

#### Public Attributes

- pdict

  *The parameter dictionary (defined in this file)*
- atom

  *The atom numbers in the interaction (stored in the parser)*
- atomnames

  *The mol2 file provides a list of atom names.*
- section

  *The section that we're in.*
- mol
- itype

- suffix
- molatom

*The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.*

#### 8.31.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 40 of file amberio.py.

#### 8.31.2 Constructor & Destructor Documentation

##### 8.31.2.1 def forcebalance.amberio.Mol2_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 42 of file amberio.py.

#### 8.31.3 Member Function Documentation

##### 8.31.3.1 def forcebalance.amberio.Mol2_Reader.feed ( *self, line* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 56 of file amberio.py.

#### 8.31.4 Member Data Documentation

##### 8.31.4.1 forcebalance::amberio.Mol2_Reader::atom

The atom numbers in the interaction (stored in the parser)

Definition at line 44 of file amberio.py.

##### 8.31.4.2 forcebalance::amberio.Mol2_Reader::atomnames

The mol2 file provides a list of atom names.

Definition at line 45 of file amberio.py.

##### 8.31.4.3 forcebalance::amberio.Mol2_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 56 of file amberio.py.

##### 8.31.4.4 forcebalance::amberio.Mol2_Reader::mol

Definition at line 46 of file amberio.py.

##### 8.31.4.5 forcebalance::amberio.Mol2_Reader::molatom

The mapping of (molecule name) to a dictionary of of atom types for the atoms in that residue.

self.moleculedict = OrderedDict() The listing of 'RES:ATOMNAMES' for atom names in the line This is obviously a placeholder.

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 56 of file amberio.py.

#### 8.31.4.6 forcebalance::amberio.Mol2_Reader::pdict

The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 43 of file amberio.py.

#### 8.31.4.7 forcebalance::amberio.Mol2_Reader::section

The section that we're in.

Definition at line 46 of file amberio.py.

#### 8.31.4.8 forcebalance::amberio.Mol2_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 56 of file amberio.py.

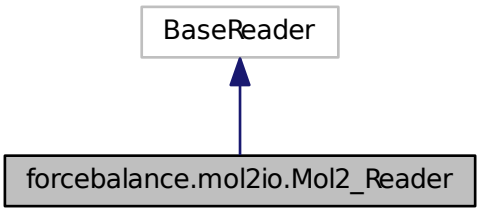
The documentation for this class was generated from the following file:

- amberio.py

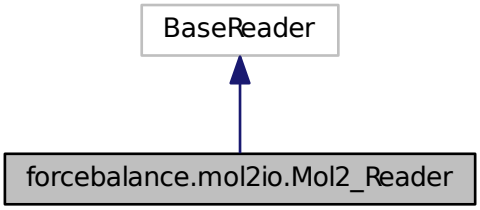
## 8.32 forcebalance.mol2io.Mol2_Reader Class Reference

Finite state machine for parsing Mol2 force field file.

Inheritance diagram for forcebalance.mol2io.Mol2_Reader:



Collaboration diagram for forcebalance.mol2io.Mol2_Reader:




#### Public Member Functions

- def __init__
- def feed

#### Public Attributes

- pdict

  *The parameter dictionary (defined in this file)*
- atom

  *The atom numbers in the interaction (stored in the parser)*
- itype
- suffix

### 8.32.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 22 of file mol2io.py.

### 8.32.2 Constructor & Destructor Documentation

#### 8.32.2.1 def forcebalance.mol2io.Mol2_Reader.__init__ ( *self, fnm* )

Definition at line 24 of file mol2io.py.

### 8.32.3 Member Function Documentation

#### 8.32.3.1 def forcebalance.mol2io.Mol2_Reader.feed ( *self, line* )

Definition at line 32 of file mol2io.py.

#### 8.32.4 Member Data Documentation

##### 8.32.4.1 forcebalance::mol2io.Mol2_Reader::atom

The atom numbers in the interaction (stored in the parser)

Definition at line 26 of file mol2io.py.

##### 8.32.4.2 forcebalance::mol2io.Mol2_Reader::itype

Definition at line 32 of file mol2io.py.

##### 8.32.4.3 forcebalance::mol2io.Mol2_Reader::pdict

The parameter dictionary (defined in this file)

Definition at line 25 of file mol2io.py.

##### 8.32.4.4 forcebalance::mol2io.Mol2_Reader::suffix

Definition at line 32 of file mol2io.py.

The documentation for this class was generated from the following file:

- mol2io.py

## 8.33 forcebalance.Mol2.mol2_set Class Reference

#### Public Member Functions

- def __init__

  *A collection is organized as a dictionnary of compounds self.num_compounds : the number of compounds self.compounds : the dictionnary of compounds data : the data to setup the set subset: it is possible to specify a subset of the compounds to load, based on their mol_name identifiers.*
- def parse

  *parse a list of lines, detect compounds, load them only load the subset if specified.*

#### Public Attributes

- num_compounds
- comments
- compounds

#### 8.33.1 Detailed Description

Definition at line 568 of file Mol2.py.

#### 8.33.2 Constructor & Destructor Documentation

##### 8.33.2.1 def forcebalance.Mol2.mol2_set.__init__ ( *self*, *data* = None, *subset* = None )

A collection is organized as a dictionnary of compounds self.num_compounds : the number of compounds self.compounds : the dictionnary of compounds data : the data to setup the set subset: it is possible to specify a subset of the compounds to load, based on their mol_name identifiers.

Definition at line 577 of file Mol2.py.

#### 8.33.3 Member Function Documentation

##### 8.33.3.1 def forcebalance.Mol2.mol2_set.parse ( *self, data, subset* = `None` )

parse a list of lines, detect compounds, load them only load the subset if specified.

Definition at line 621 of file Mol2.py.

#### 8.33.4 Member Data Documentation

##### 8.33.4.1 forcebalance::Mol2.mol2_set::comments

Definition at line 577 of file Mol2.py.

##### 8.33.4.2 forcebalance::Mol2.mol2_set::compounds

Definition at line 577 of file Mol2.py.

##### 8.33.4.3 forcebalance::Mol2.mol2_set::num_compounds

Definition at line 577 of file Mol2.py.

The documentation for this class was generated from the following file:

- Mol2.py

## 8.34 forcebalance.molecule.Molecule Class Reference

Lee-Ping's general file format conversion class.

#### Public Member Functions

- def __len__
  *Return the number of frames in the trajectory.*
- def __getattr__
  *Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.*
- def __setattr__
  *Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.*
- def __getitem__
  *The Molecule class has list-like behavior, so we can get slices of it.*
- def __delitem__
  *Similarly, in order to delete a frame, we simply perform item deletion on framewise variables.*
- def __iter__
  *List-like behavior for looping over trajectories.*
- def __add__
  *Add method for Molecule objects.*
- def __iadd__
  *Add method for Molecule objects.*

- def append
- def __init__

  *To create the Molecule object, we simply define the table of file reading/writing functions and read in a file if it is provided.*
- def require
- def write
- def center_of_mass
- def radius_of_gyration
- def load_frames
- def edit_qcrems
- def add_quantum
- def add_virtual_site

  *Add a virtual site to the system.*
- def replace_peratom

  *Replace all of the data for a certain attribute in the system from orig to want.*
- def replace_peratom_conditional

  *Replace all of the data for a attribute key2 from orig to want, contingent on key1 being equal to cond.*
- def atom_select

  *Return a copy of the object with certain atoms selected.*
- def atom_stack

  *Return a copy of the object with another molecule object appended.*
- def align_by_moments

  *Align molecules using the "moment of inertia." Note that we're following the MSMBuilder convention of using all ones for the masses.*
- def align

  *Align molecules.*
- def build_topology

  *A bare-bones implementation of the bond graph capability in the nanoreactor code.*
- def measure_dihedrals

  *Return a series of dihedral angles, given four atom indices numbered from zero.*
- def all_pairwise_rmsd

  *Find pairwise RMSD (super slow, not like the one in MSMBuilder.)*
- def align_center
- def openmm_positions

  *Returns the Cartesian coordinates in the Molecule object in a list of OpenMM-compatible positions, so it is possible to type simulation.context.setPositions(Mol.openmm_positions()[0]) or something like that.*
- def openmm_boxes

  *Returns the periodic box vectors in the Molecule object in a list of OpenMM-compatible boxes, so it is possible to type simulation.context.setPeriodicBoxVectors(Mol.openmm_boxes()[0]) or something like that.*
- def split

  *Split the molecule object into a number of separate files (chunks), either by specifying the number of frames per chunk or the number of chunks.*
- def read_xyz

  *Parse a .xyz file which contains several xyz coordinates, and return their elements.*
- def read_mdcrd

  *Parse an AMBER .mdcrd file.*
- def read_qdata
- def read_mol2
- def read_dcd

- def read_com

  *Parse a Gaussian .com file and return a SINGLE-ELEMENT list of xyz coordinates (no multiple file support)*
- def read_arc

  *Read a TINKER .arc file.*
- def read_gro

  *Read a GROMACS .gro file.*
- def read_charmm

  *Read a CHARMM .cor (or .crd) file.*
- def read_qcin

  *Read a Q-Chem input file.*
- def read_pdb

  *Loads a PDB and returns a dictionary containing its data.*
- def read_qcesp
- def read_qcout

  *Q-Chem output file reader, adapted for our parser.*
- def write_qcin
- def write_xyz
- def write_molproq
- def write_mdcrd
- def write_arc
- def write_gro
- def write_dcd
- def write_pdb

  *Save to a PDB.*
- def write_qdata

  *Text quantum data format.*
- def require_resid
- def require_resname
- def require_boxes

**Public Attributes**

- Read_Tab

  *The table of file readers.*
- Write_Tab

  *The table of file writers.*
- Funnel

  *A funnel dictionary that takes redundant file types and maps them down to a few.*
- positive_resid

  *Creates entries like 'gromacs' : 'gromacs' and 'xyz' : 'xyz' in the Funnel.*
- Data
- comms

  *Read in stuff if we passed in a file name, otherwise return an empty instance.*
- topology

  *Make sure the comment line isn't too long for i in range(len(self.comms)): self.comms[i] = self.comms[i][:100] if len(self.comms[i]) > 100 else self.comms[i] Attempt to build the topology for small systems.*
- molecules
- fout

*Fill in comments.*

- resid
- resname
- boxes

#### 8.34.1 Detailed Description

Lee-Ping's general file format conversion class.

The purpose of this class is to read and write chemical file formats in a way that is convenient for research. There are highly general file format converters out there (e.g. catdcd, openbabel) but I find that writing my own class can be very helpful for specific purposes. Here are some things this class can do:

- Convert a .gro file to a .xyz file, or a .pdb file to a .dcd file. Data is stored internally, so any readable file can be converted into any writable file as long as there is sufficient information to write that file.
- Accumulate information from different files. For example, we may read A.gro to get a list of coordinates, add quantum settings from a B.in file, and write A.in (this gives us a file that we can use to run QM calculations)
- Concatenate two trajectories together as long as they're compatible. This is done by creating two Molecule objects and then simply adding them. Addition means two things: (1) Information fields missing from each class, but present in the other, are added to the sum, and (2) Appendable or per-frame fields (i.e. coordinates) are concatenated together.
- Slice trajectories using reasonable Python language. That is to say, MyMolecule[1:10] returns a new Molecule object that contains frames 2 through 10.

Next step: Read in Q-Chem output data using this too!

Special variables: These variables cannot be set manually because there is a special method associated with getting them.

na = The number of atoms. You'll get this if you use MyMol.na or MyMol['na']. na = The number of snapshots. You'll get this if you use MyMol.ns or MyMol['ns'].

Unit system: Angstroms.

Definition at line 647 of file molecule.py.

#### 8.34.2 Constructor & Destructor Documentation

##### 8.34.2.1 def forcebalance.molecule.Molecule.__init__ ( *self*, *fnm* = `None`, *ftype* = `None`, *positive_resid* = `True`, *build_topology* = `True` )

To create the Molecule object, we simply define the table of file reading/writing functions and read in a file if it is provided.

Definition at line 817 of file molecule.py.

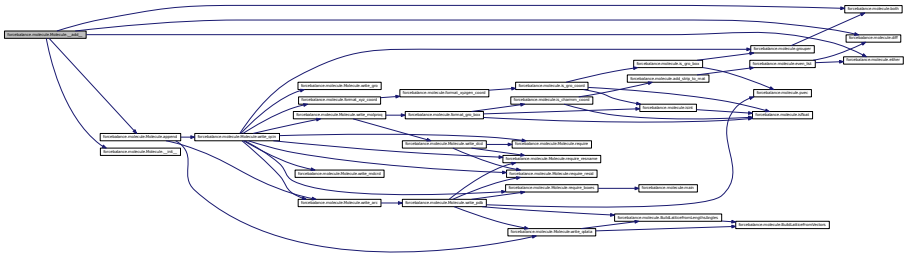
#### 8.34.3 Member Function Documentation

##### 8.34.3.1 def forcebalance.molecule.Molecule.__add__ ( *self*, *other* )

Add method for Molecule objects.

Definition at line 756 of file molecule.py.

Here is the call graph for this function:

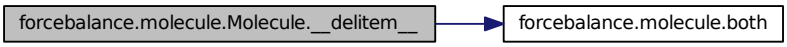


#### 8.34.3.2 def forcebalance.molecule.Molecule.__delitem__ ( self, key )

Similarly, in order to delete a frame, we simply perform item deletion on framewise variables.

Definition at line 736 of file molecule.py.

Here is the call graph for this function:

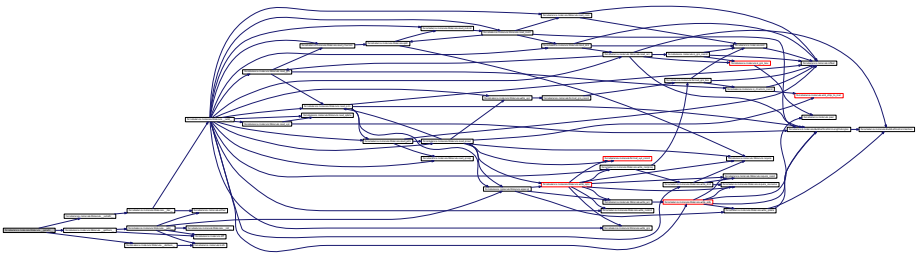



#### 8.34.3.3 def forcebalance.molecule.Molecule.__getattr__ ( self, key )

Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.

Definition at line 667 of file molecule.py.

Here is the call graph for this function:



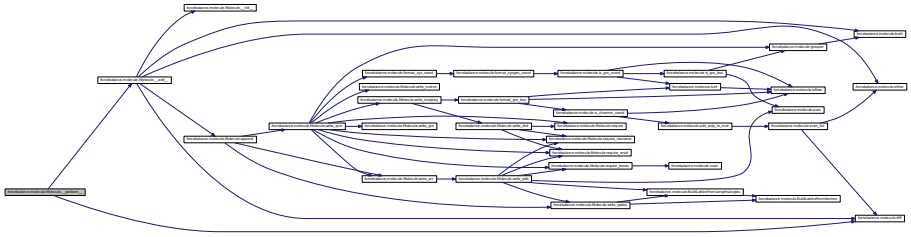
#### 8.34.3.4 def forcebalance.molecule.Molecule.__getitem__ ( self, key )

The Molecule class has list-like behavior, so we can get slices of it.

If we say MyMolecule[0:10], then we'll return a copy of MyMolecule with frames 0 through 9.

Definition at line 715 of file molecule.py.

Here is the call graph for this function:

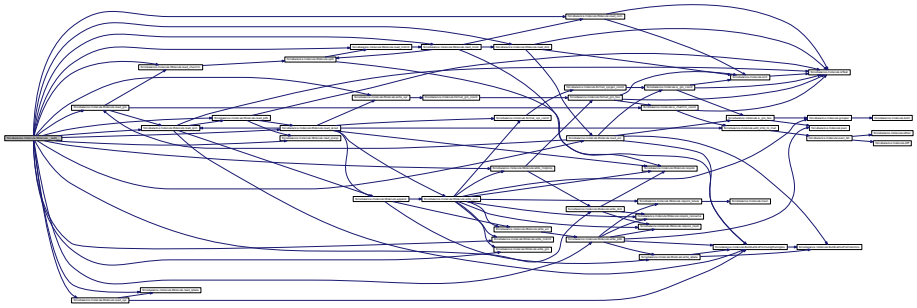


#### 8.34.3.5 def forcebalance.molecule.Molecule.__iadd__ ( *self, other* )

Add method for Molecule objects.

Definition at line 786 of file molecule.py.

Here is the call graph for this function:



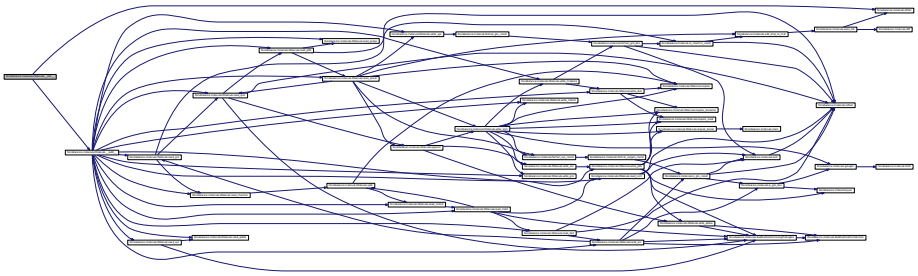
#### 8.34.3.6 def forcebalance.molecule.Molecule.__iter__ ( *self* )

List-like behavior for looping over trajectories.

Note that these values are returned by reference. Note that this is intended to be more efficient than __getitem__, so when we loop over a trajectory, it's best to go "for m in M" instead of "for i in range(len(M)): m = M[i]"

Definition at line 745 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.7 def forcebalance.molecule.Molecule.__len__ ( *self* )

Return the number of frames in the trajectory.

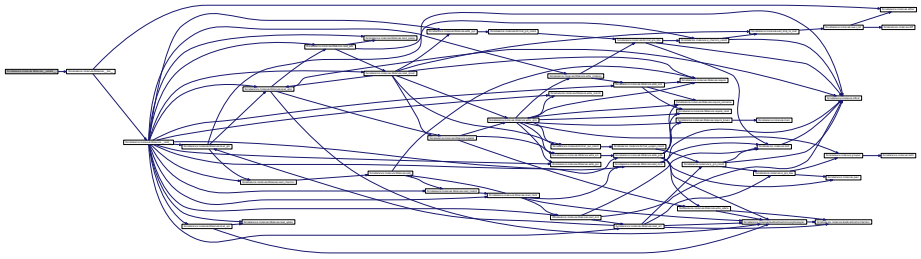
Definition at line 651 of file molecule.py.

#### 8.34.3.8 def forcebalance.molecule.Molecule.__setattr__ ( *self, key, value* )

Whenever we try to get a class attribute, it first tries to get the attribute from the Data dictionary.

Definition at line 703 of file molecule.py.

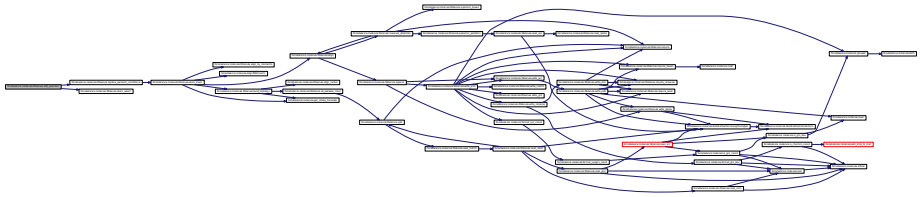
Here is the call graph for this function:



#### 8.34.3.9 def forcebalance.molecule.Molecule.add_quantum ( *self, other* )

Definition at line 997 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.10 def forcebalance.molecule.Molecule.add_virtual_site ( *self, idx, kwargs* )

Add a virtual site to the system.

This does NOT set the position of the virtual site; it sits at the origin.

Definition at line 1009 of file molecule.py.

#### 8.34.3.11 def forcebalance.molecule.Molecule.align ( *self, smooth* = `True`, *center* = `True` )

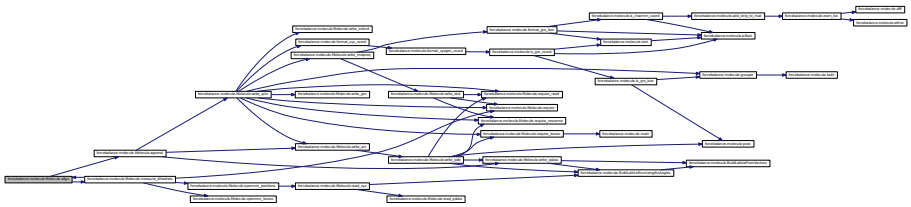
Align molecules.

Has the option to create smooth trajectories (align each frame to the previous one) or to align each frame to the first one.

Also has the option to remove the center of mass.

Definition at line 1145 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.12 def forcebalance.molecule.Molecule.align_by_moments ( *self* )

Align molecules using the "moment of inertia." Note that we're following the MSMBuilder convention of using all ones for the masses.

Definition at line 1126 of file molecule.py.

#### 8.34.3.13 def forcebalance.molecule.Molecule.align_center ( *self* )

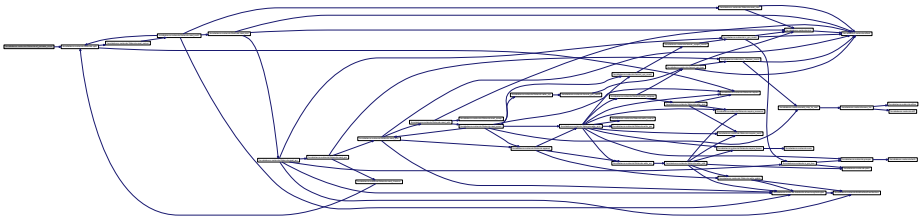
Definition at line 1249 of file molecule.py.

#### 8.34.3.14 def forcebalance.molecule.Molecule.all_pairwise_rmsd ( *self* )

Find pairwise RMSD (super slow, not like the one in MSMBuilder.)

Definition at line 1233 of file molecule.py.

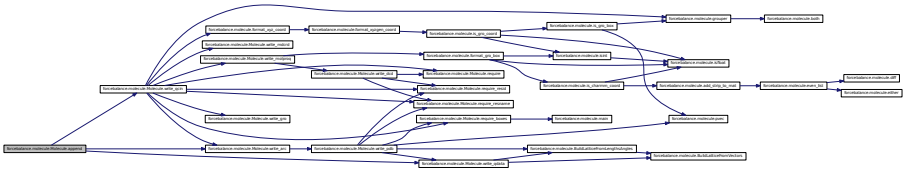
Here is the call graph for this function:



#### 8.34.3.15 def forcebalance.molecule.Molecule.append ( *self, other* )

Definition at line 810 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.16 def forcebalance.molecule.Molecule.atom_select ( *self, atomslice* )

Return a copy of the object with certain atoms selected.

Takes an integer, list or array as argument.

Definition at line 1047 of file molecule.py.

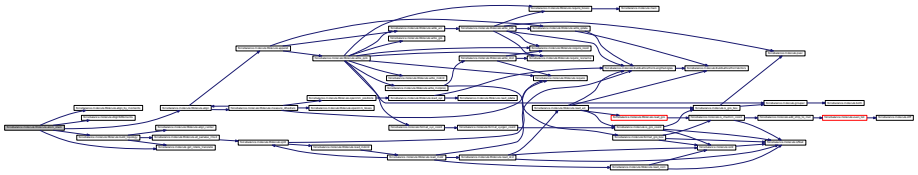
#### 8.34.3.17 def forcebalance.molecule.Molecule.atom_stack ( *self, other* )

Return a copy of the object with another molecule object appended.

WARNING: This function may invalidate stuff like QM energies.

Definition at line 1082 of file molecule.py.

Here is the call graph for this function:



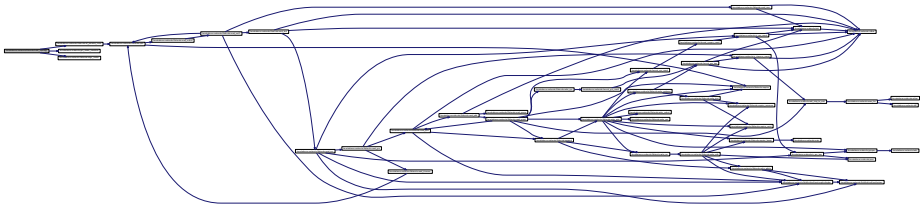
#### 8.34.3.18 def forcebalance.molecule.Molecule.build_topology ( *self, sn* = `None`, *Fac* = `1.2` )

A bare-bones implementation of the bond graph capability in the nanoreactor code.

Returns a NetworkX graph that depicts the molecular topology, which might be useful for stuff. Provide, optionally, the frame number used to compute the topology.

Definition at line 1164 of file molecule.py.

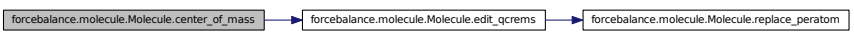
Here is the call graph for this function:



#### 8.34.3.19 def forcebalance.molecule.Molecule.center_of_mass ( *self* )

Definition at line 954 of file molecule.py.

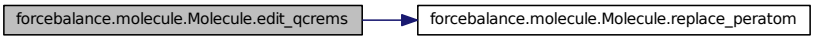
Here is the call graph for this function:



#### 8.34.3.20 def forcebalance.molecule.Molecule.edit_qcrems ( *self, in_dict, subcalc* = None )

Definition at line 988 of file molecule.py.

Here is the call graph for this function:




#### 8.34.3.21 def forcebalance.molecule.Molecule.load_frames ( *self, fnm* )

Definition at line 981 of file molecule.py.

#### 8.34.3.22 def forcebalance.molecule.Molecule.measure_dihedrals ( *self, i, j, k, l* )

Return a series of dihedral angles, given four atom indices numbered from zero.

Definition at line 1206 of file molecule.py.

Here is the call graph for this function:

#### 8.34.3.23 def forcebalance.molecule.Molecule.openmm_boxes ( *self* )

Returns the periodic box vectors in the Molecule object in a list of OpenMM-compatible boxes, so it is possible to type simulation.context.setPeriodicBoxVectors(Mol.openmm_boxes()[0]) or something like that.

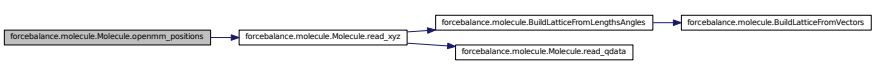
Definition at line 1275 of file molecule.py.

#### 8.34.3.24 def forcebalance.molecule.Molecule.openmm_positions ( *self* )

Returns the Cartesian coordinates in the Molecule object in a list of OpenMM-compatible positions, so it is possible to type simulation.context.setPositions(Mol.openmm_positions()[0]) or something like that.

Definition at line 1258 of file molecule.py.

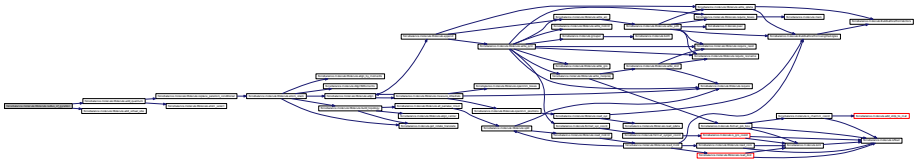
Here is the call graph for this function:



#### 8.34.3.25 def forcebalance.molecule.Molecule.radius_of_gyration ( *self* )

Definition at line 958 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.26 def forcebalance.molecule.Molecule.read_arc ( *self, fnm* )

Read a TINKER .arc file.

**Parameters**

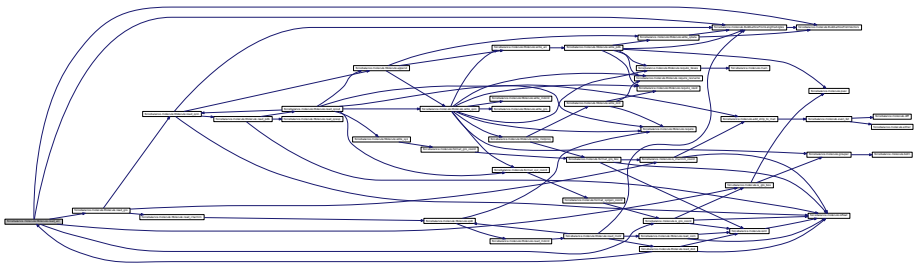
| | | |
|---|---|---|
| in | *fnm* | The input file name |

**Returns**

xyzs A list for the XYZ coordinates.
resid The residue ID numbers. These are not easy to get!
elem A list of chemical elements in the XYZ file
comms A single-element list for the comment.
tinkersuf The suffix that comes after lines in the XYZ coordinates; this is usually topology info

Definition at line 1552 of file molecule.py.

Here is the call graph for this function:

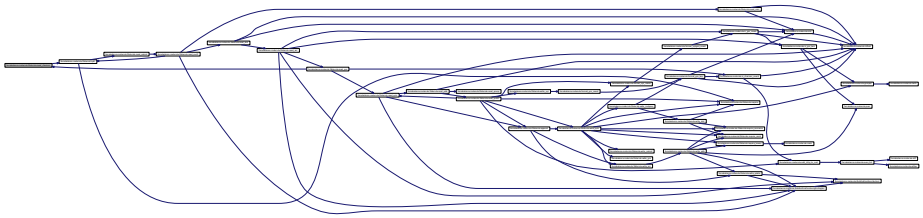


#### 8.34.3.27 def forcebalance.molecule.Molecule.read_charmm ( *self, fnm* )

Read a CHARMM .cor (or .crd) file.

Definition at line 1679 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.28 def forcebalance.molecule.Molecule.read_com ( *self, fnm* )

Parse a Gaussian .com file and return a SINGLE-ELEMENT list of xyz coordinates (no multiple file support)

**Parameters**

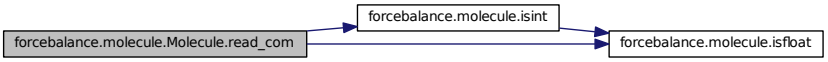
| | | |
|---|---|---|
| in | *fnm* | The input file name |

**Returns**

elem A list of chemical elements in the XYZ file
comms A single-element list for the comment.
xyzs A single-element list for the XYZ coordinates.
charge The total charge of the system.
mult The spin multiplicity of the system.

Definition at line 1509 of file molecule.py.

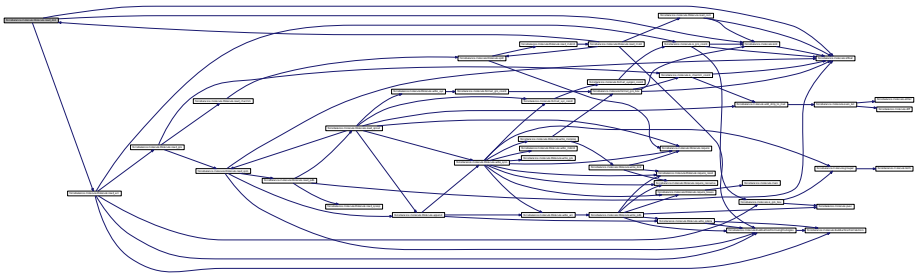
Here is the call graph for this function:




#### 8.34.3.29 def forcebalance.molecule.Molecule.read_dcd ( *self, fnm* )

Definition at line 1470 of file molecule.py.

Here is the call graph for this function:

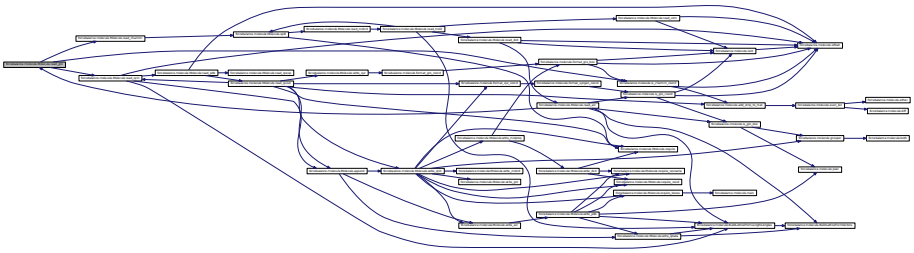


#### 8.34.3.30 def forcebalance.molecule.Molecule.read_gro ( *self, fnm* )

Read a GROMACS .gro file.

Definition at line 1615 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.31 def forcebalance.molecule.Molecule.read_mdcrd ( *self, fnm* )

Parse an AMBER .mdcrd file.

This requires at least the number of atoms. This will FAIL for monatomic trajectories (but who the heck makes those?)

**Parameters**

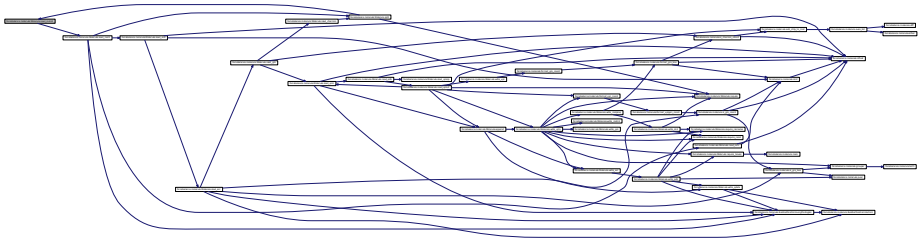
| | | |
|---|---|---|
| in | *fnm* | The input file name |

**Returns**

xyzs A list of XYZ coordinates (number of snapshots times number of atoms)
boxes Boxes (if present.)

Definition at line 1361 of file molecule.py.

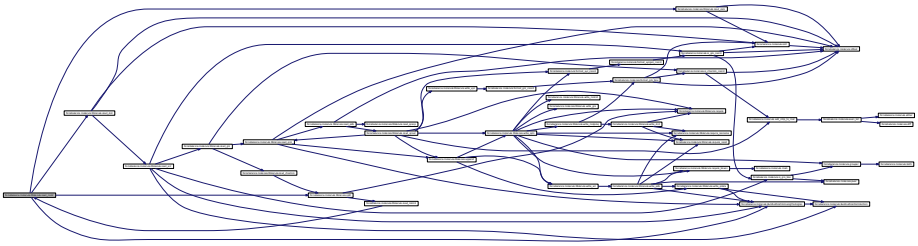
Here is the call graph for this function:



#### 8.34.3.32 def forcebalance.molecule.Molecule.read_mol2 ( *self, fnm* )

Definition at line 1421 of file molecule.py.

Here is the call graph for this function:

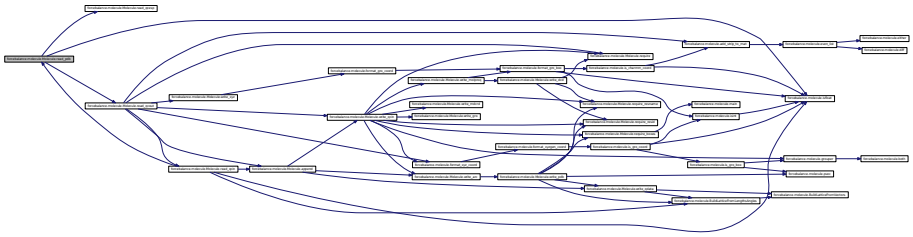


#### 8.34.3.33 def forcebalance.molecule.Molecule.read_pdb ( *self, fnm* )

Loads a PDB and returns a dictionary containing its data.

Definition at line 1859 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.34 def forcebalance.molecule.Molecule.read_qcesp ( *self, fnm* )

Definition at line 1934 of file molecule.py.

#### 8.34.3.35 def forcebalance.molecule.Molecule.read_qcin ( *self, fnm* )

Read a Q-Chem input file.

These files can be very complicated, and I can't write a completely general parser for them. It is important to keep our goal in mind:

1) The main goal is to convert a trajectory to Q-Chem input files with identical calculation settings.

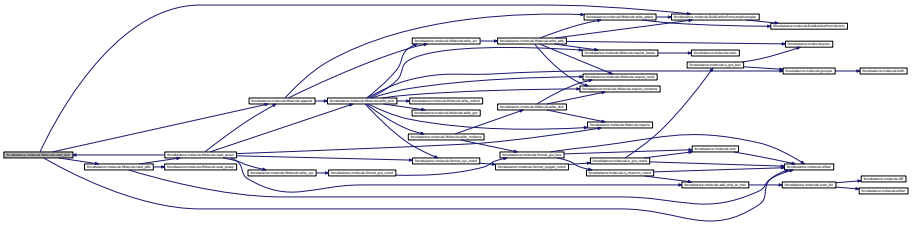
2) When we print the Q-Chem file, we should preserve the line ordering of the 'rem' section, but also be able to add 'rem' options at the end.

3) We should accommodate the use case that the Q-Chem file may have follow-up calculations delimited by '@@'.

4) We can read in all of the xyz's as a trajectory, but only the Q-Chem settings belonging to the first xyz will be saved.

Definition at line 1747 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.36 def forcebalance.molecule.Molecule.read_qcout ( *self, fnm* )

Q-Chem output file reader, adapted for our parser.

Q-Chem output files are very flexible and there's no way I can account for all of them. Here's what I am able to account for:

A list of:

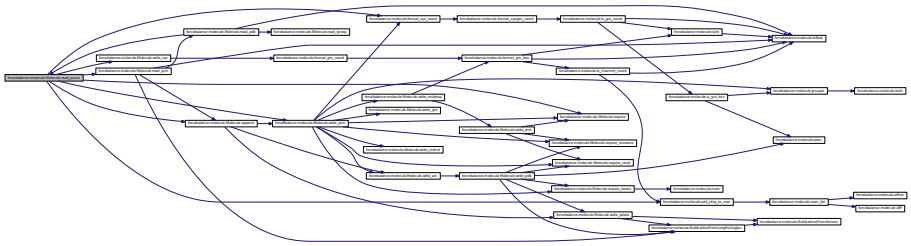
- Coordinates
- Energies
- Forces

Note that each step in a geometry optimization counts as a frame.

As with all Q-Chem output files, note that successive calculations can have different numbers of atoms.

Definition at line 1963 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.37 def forcebalance.molecule.Molecule.read_qdata ( *self, fnm* )

Definition at line 1386 of file molecule.py.

#### 8.34.3.38 def forcebalance.molecule.Molecule.read_xyz ( *self, fnm* )

Parse a .xyz file which contains several xyz coordinates, and return their elements.

**Parameters**

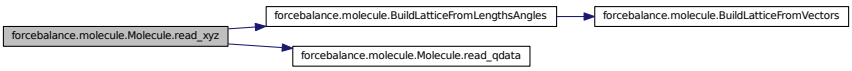
| | | |
|---|---|---|
| in | *fnm* | The input file name |

**Returns**

elem A list of chemical elements in the XYZ file
comms A list of comments.
xyzs A list of XYZ coordinates (number of snapshots times number of atoms)

Definition at line 1316 of file molecule.py.

Here is the call graph for this function:




#### 8.34.3.39 def forcebalance.molecule.Molecule.replace_peratom ( *self, key, orig, want* )

Replace all of the data for a certain attribute in the system from orig to want.

Definition at line 1026 of file molecule.py.

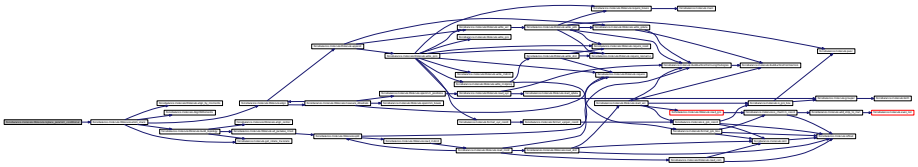
#### 8.34.3.40 def forcebalance.molecule.Molecule.replace_peratom_conditional ( *self, key1, cond, key2, orig, want* )

Replace all of the data for a attribute key2 from orig to want, contingent on key1 being equal to cond.

For instance: replace H1 with H2 if resname is SOL.

Definition at line 1037 of file molecule.py.

Here is the call graph for this function:



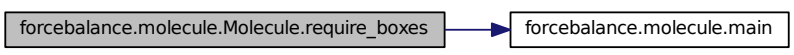
#### 8.34.3.41 def forcebalance.molecule.Molecule.require ( *self, args* )

Definition at line 905 of file molecule.py.

#### 8.34.3.42 def forcebalance.molecule.Molecule.require_boxes ( *self* )

Definition at line 2436 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.43 def forcebalance.molecule.Molecule.require_resid ( *self* )

Definition at line 2423 of file molecule.py.

#### 8.34.3.44 def forcebalance.molecule.Molecule.require_resname ( *self* )

Definition at line 2431 of file molecule.py.

#### 8.34.3.45 def forcebalance.molecule.Molecule.split ( *self, fnm* = None, *ftype* = None, *method* = "chunks", *num* = None )

Split the molecule object into a number of separate files (chunks), either by specifying the number of frames per chunk or the number of chunks.

Only relevant for "trajectories". The type of file may be specified; if they aren't specified then the original file type is used.

The output file names are [name].[numbers].[extension] where [name] can be specified by passing 'fnm' or taken from the object's 'fnm' attribute by default. [numbers] are integers ranging from the lowest to the highest chunk number, prepended by zeros.

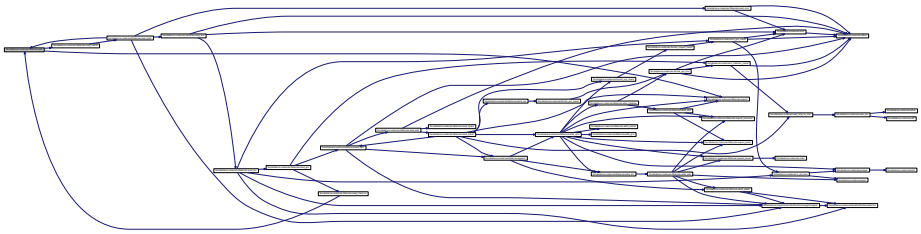
If the number of chunks / frames is not specified, then one file is written for each frame.

**Returns**

fnms A list of the file names that were written.

Definition at line 1298 of file molecule.py.

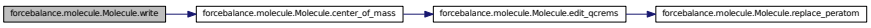
Here is the call graph for this function:



#### 8.34.3.46 def forcebalance.molecule.Molecule.write ( *self, fnm* = None, *ftype* = None, *append* = False, *select* = None )

Definition at line 920 of file molecule.py.

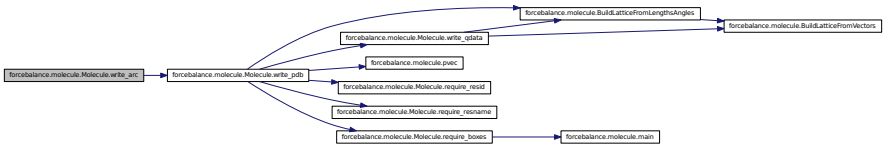
Here is the call graph for this function:




#### 8.34.3.47 def forcebalance.molecule.Molecule.write_arc ( *self, select* )

Definition at line 2220 of file molecule.py.

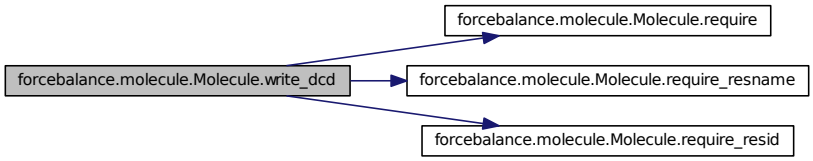
Here is the call graph for this function:




#### 8.34.3.48 def forcebalance.molecule.Molecule.write_dcd ( *self, select* )

Definition at line 2256 of file molecule.py.

Here is the call graph for this function:




#### 8.34.3.49 def forcebalance.molecule.Molecule.write_gro ( *self, select* )

Definition at line 2232 of file molecule.py.

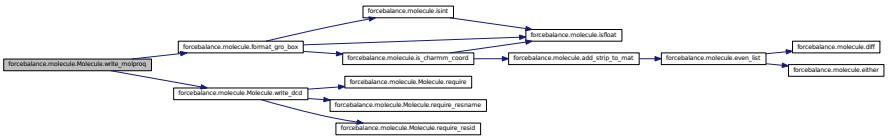
#### 8.34.3.50 def forcebalance.molecule.Molecule.write_mdcrd ( *self, select* )

Definition at line 2209 of file molecule.py.

#### 8.34.3.51 def forcebalance.molecule.Molecule.write_molproq ( *self, select* )

Definition at line 2197 of file molecule.py.

Here is the call graph for this function:




#### 8.34.3.52 def forcebalance.molecule.Molecule.write_pdb ( *self, select* )

Save to a PDB.

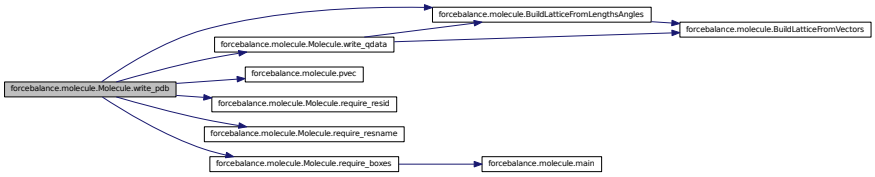
Copied wholesale from MSMBuilder. COLUMNS TYPE FIELD DEFINITION ---------------------------------------------- 7-11 int serial Atom serial number. 13-16 string name Atom name. 17 string altLoc Alternate location indicator. 18-20 (17-21

KAB) string resName Residue name. 22 string chainID Chain identifier. 23-26 int resSeq Residue sequence number. 27 string iCode Code for insertion of residues. 31-38 float x Orthogonal coordinates for X in Angstroms. 39-46 float y Orthogonal coordinates for Y in Angstroms. 47-54 float z Orthogonal coordinates for Z in Angstroms. 55-60 float occupancy Occupancy. 61-66 float tempFactor Temperature factor. 73-76 string segID Segment identifier, left-justified. 77-78 string element Element symbol, right-justified. 79-80 string charge Charge on the atom.

CRYST1 line, added by Lee-Ping COLUMNS TYPE FIELD DEFINITION ---------- 7-15 float a a (Angstroms). 16-24 float b b (Angstroms). 25-33 float c c (Angstroms). 34-40 float alpha alpha (degrees). 41-47 float beta beta (degrees). 48-54 float gamma gamma (degrees). 56-66 string sGroup Space group. 67-70 int z Z value.

Definition at line 2312 of file molecule.py.

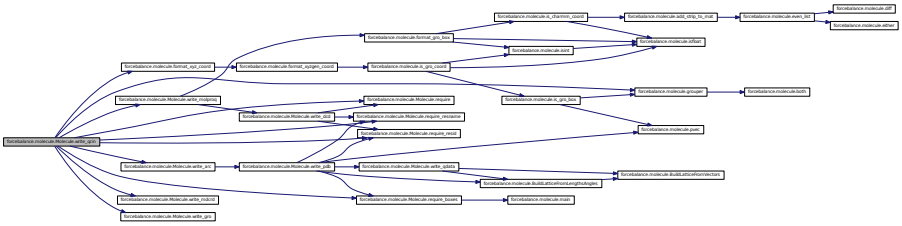
Here is the call graph for this function:




#### 8.34.3.53 def forcebalance.molecule.Molecule.write_qcin ( *self, select* )

Definition at line 2140 of file molecule.py.

Here is the call graph for this function:



#### 8.34.3.54 def forcebalance.molecule.Molecule.write_qdata ( *self, select* )

Text quantum data format.

Definition at line 2402 of file molecule.py.

Here is the call graph for this function:


#### 8.34.3.55 def forcebalance.molecule.Molecule.write_xyz ( *self, select* )

Definition at line 2186 of file molecule.py.

Here is the call graph for this function:


#### 8.34.4 Member Data Documentation

##### 8.34.4.1 forcebalance::molecule.Molecule::boxes

Definition at line 2436 of file molecule.py.

##### 8.34.4.2 forcebalance::molecule.Molecule::comms

Read in stuff if we passed in a file name, otherwise return an empty instance.

Try to determine from the file name using the extension. Actually read the file. Set member variables. Create a list of comment lines if we don't already have them from reading the file.

Definition at line 828 of file molecule.py.

##### 8.34.4.3 forcebalance::molecule.Molecule::Data

Definition at line 823 of file molecule.py.

##### 8.34.4.4 forcebalance::molecule.Molecule::fout

Fill in comments.

I needed to add in this line because the DCD writer requires the file name, but the other methods don't.

Definition at line 923 of file molecule.py.

##### 8.34.4.5 forcebalance::molecule.Molecule::Funnel

A funnel dictionary that takes redundant file types and maps them down to a few.

Definition at line 821 of file molecule.py.

##### 8.34.4.6 forcebalance::molecule.Molecule::molecules

Definition at line 832 of file molecule.py.

##### 8.34.4.7 forcebalance::molecule.Molecule::positive_resid

Creates entries like 'gromacs' : 'gromacs' and 'xyz' : 'xyz' in the Funnel.

Definition at line 823 of file molecule.py.

##### 8.34.4.8 forcebalance::molecule.Molecule::Read_Tab

The table of file readers.

Definition at line 818 of file molecule.py.

##### 8.34.4.9 forcebalance::molecule.Molecule::resid

Definition at line 2423 of file molecule.py.

#### 8.34.4.10 forcebalance::molecule.Molecule::resname

Definition at line 2431 of file molecule.py.

#### 8.34.4.11 forcebalance::molecule.Molecule::topology

Make sure the comment line isn't too long for i in range(len(self.comms)): self.comms[i] = self.comms[i][:100] if len(self.comms[i]) $>$ 100 else self.comms[i] Attempt to build the topology for small systems.

:)

Definition at line 832 of file molecule.py.

#### 8.34.4.12 forcebalance::molecule.Molecule::Write_Tab

The table of file writers.

Definition at line 819 of file molecule.py.

The documentation for this class was generated from the following file:

- molecule.py

## 8.35 forcebalance.molecule.MolfileTimestep Class Reference

Wrapper for the timestep C structure used in molfile plugins.

### 8.35.1 Detailed Description

Wrapper for the timestep C structure used in molfile plugins.

Definition at line 471 of file molecule.py.

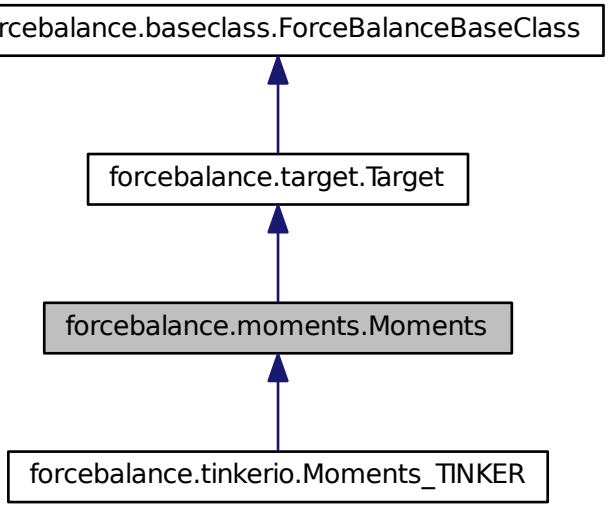
The documentation for this class was generated from the following file:

- molecule.py

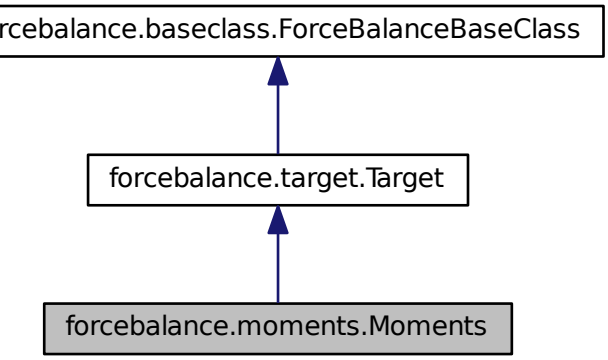
## 8.36 forcebalance.moments.Moments Class Reference

Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Inheritance diagram for forcebalance.moments.Moments:




Collaboration diagram for forcebalance.moments.Moments:




**Public Member Functions**

- def __init__

  *Initialization.*
- def read_reference_data

*Read the reference data from a file.*

- def prepare_temp_directory

  *Prepare the temporary directory.*

- def indicate

  *Print qualitative indicator.*

- def unpack_moments
- def get

  *Evaluate objective function.*

#### Public Attributes

- denoms
- mfnm

  *The mdata.txt file that contains the moments.*

- ref_moments
- na

  *Number of atoms.*

- ref_eigvals
- ref_eigvecs
- calc_moments
- objective

### 8.36.1 Detailed Description

Subclass of Target for fitting force fields to multipole moments (from experiment or theory).

Currently Tinker is supported.

Definition at line 27 of file moments.py.

### 8.36.2 Constructor & Destructor Documentation

#### 8.36.2.1 def forcebalance.moments.Moments.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization.

Reimplemented from forcebalance.target.Target.

Reimplemented in forcebalance.tinkerio.Moments_TINKER.

Definition at line 32 of file moments.py.

### 8.36.3 Member Function Documentation

#### 8.36.3.1 def forcebalance.moments.Moments.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Evaluate objective function.

Reimplemented from forcebalance.target.Target.

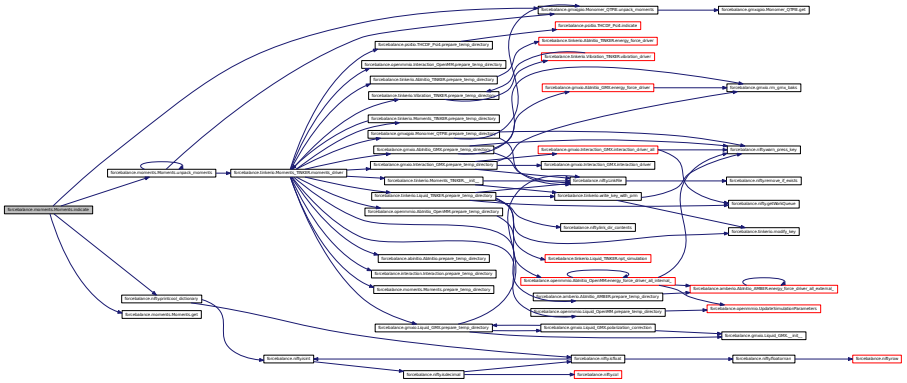
Definition at line 170 of file moments.py.

#### 8.36.3.2 def forcebalance.moments.Moments.indicate ( *self* )

Print qualitative indicator.

Definition at line 138 of file moments.py.

Here is the call graph for this function:



#### 8.36.3.3 def forcebalance.moments.Moments.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory.

Reimplemented in forcebalance.tinkerio.Moments_TINKER.

Definition at line 133 of file moments.py.

#### 8.36.3.4 def forcebalance.moments.Moments.read_reference_data ( *self* )

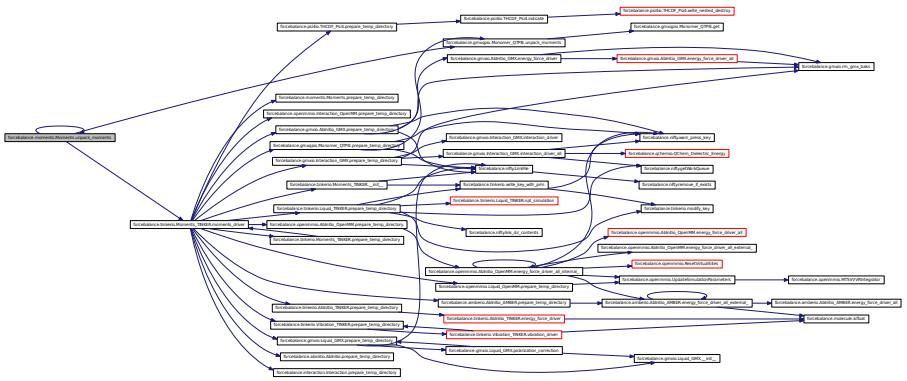
Read the reference data from a file.

Definition at line 64 of file moments.py.

#### 8.36.3.5 def forcebalance.moments.Moments.unpack_moments ( *self, moment_dict* )

Definition at line 164 of file moments.py.

Here is the call graph for this function:



#### 8.36.4 Member Data Documentation

##### 8.36.4.1 forcebalance::moments.Moments::calc_moments

Definition at line 170 of file moments.py.

##### 8.36.4.2 forcebalance::moments.Moments::denoms

Definition at line 32 of file moments.py.

##### 8.36.4.3 forcebalance::moments.Moments::mfnm

The mdata.txt file that contains the moments.

Definition at line 33 of file moments.py.

##### 8.36.4.4 forcebalance::moments.Moments::na

Number of atoms.

Definition at line 65 of file moments.py.

##### 8.36.4.5 forcebalance::moments.Moments::objective

Definition at line 170 of file moments.py.

##### 8.36.4.6 forcebalance::moments.Moments::ref_eigvals

Definition at line 65 of file moments.py.

##### 8.36.4.7 forcebalance::moments.Moments::ref_eigvecs

Definition at line 65 of file moments.py.

##### 8.36.4.8 forcebalance::moments.Moments::ref_moments

Definition at line 33 of file moments.py.

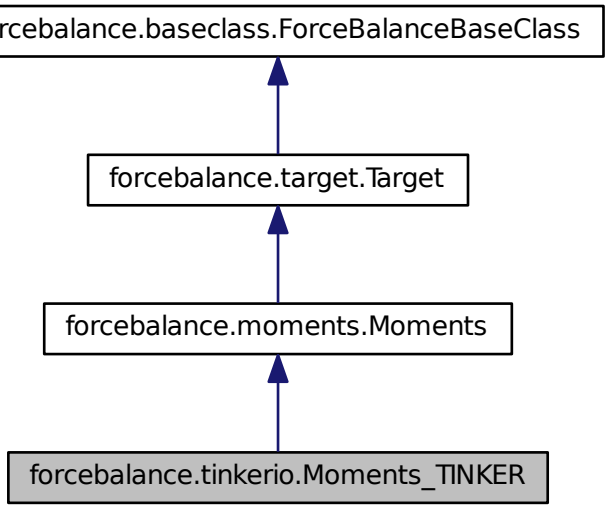
The documentation for this class was generated from the following file:

- moments.py

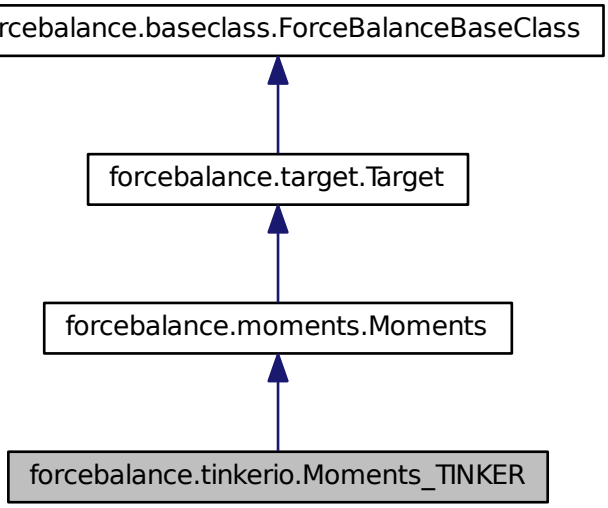
## 8.37 forcebalance.tinkerio.Moments_TINKER Class Reference

Subclass of Target for multipole moment matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Moments_TINKER:




Collaboration diagram for forcebalance.tinkerio.Moments_TINKER:



#### Public Member Functions

- def __init__
  *Initialization.*
- def prepare_temp_directory
  *Prepare the temporary directory.*
- def moments_driver

### 8.37.1 Detailed Description

Subclass of Target for multipole moment matching using TINKER.

Definition at line 402 of file tinkerio.py.

### 8.37.2 Constructor & Destructor Documentation

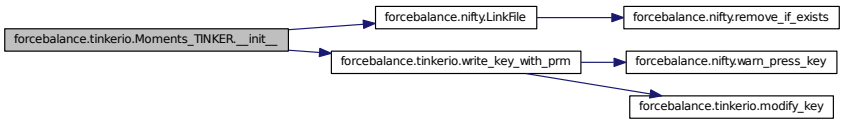
#### 8.37.2.1 def forcebalance.tinkerio.Moments_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization.

Reimplemented from forcebalance.moments.Moments.

Definition at line 405 of file tinkerio.py.

Here is the call graph for this function:

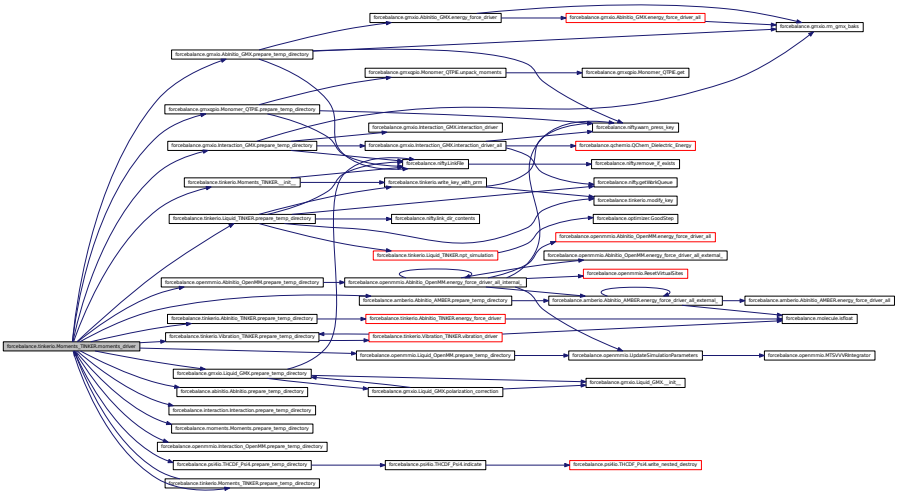



### 8.37.3 Member Function Documentation

#### 8.37.3.1 def forcebalance.tinkerio.Moments_TINKER.moments_driver ( *self* )

Definition at line 421 of file tinkerio.py.

Here is the call graph for this function:



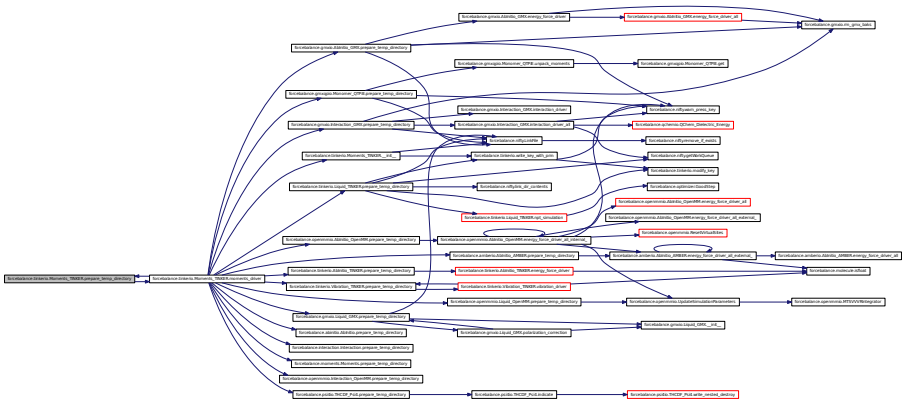
#### 8.37.3.2 def forcebalance.tinkerio.Moments_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory.

Reimplemented from forcebalance.moments.Moments.

Definition at line 410 of file tinkerio.py.

Here is the call graph for this function:



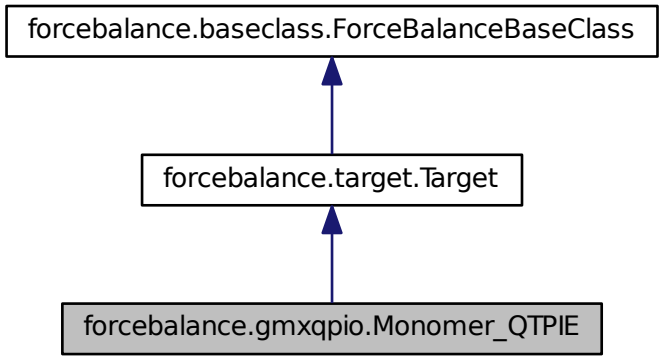
The documentation for this class was generated from the following file:

- tinkerio.py

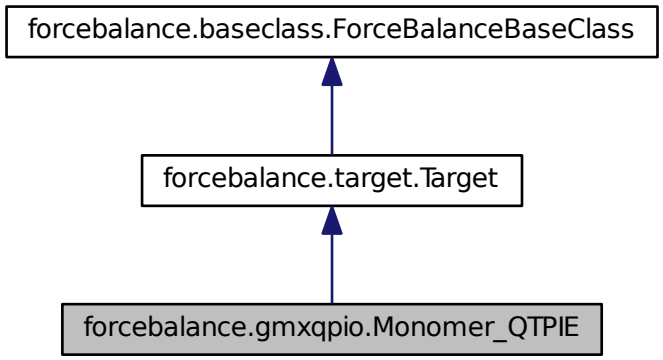
## 8.38 forcebalance.gmxqpio.Monomer_QTPIE Class Reference

Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Inheritance diagram for forcebalance.gmxqpio.Monomer_QTPIE:




Collaboration diagram for forcebalance.gmxqpio.Monomer_QTPIE:




**Public Member Functions**

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def indicate

  *Print qualitative indicator.*
- def prepare_temp_directory
- def unpack_moments

- def get

  *Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.*

**Public Attributes**

- ref_moments
- weights
- calc_moments
- objective

### 8.38.1 Detailed Description

Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Definition at line 127 of file gmxqpio.py.

### 8.38.2 Constructor & Destructor Documentation

#### 8.38.2.1 def forcebalance.gmxqpio.Monomer_QTPIE.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.target.Target.

Definition at line 129 of file gmxqpio.py.

### 8.38.3 Member Function Documentation

#### 8.38.3.1 def forcebalance.gmxqpio.Monomer_QTPIE.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.

See abinitio for an example.

Reimplemented from forcebalance.target.Target.

Definition at line 218 of file gmxqpio.py.

#### 8.38.3.2 def forcebalance.gmxqpio.Monomer_QTPIE.indicate ( *self* )

Print qualitative indicator.

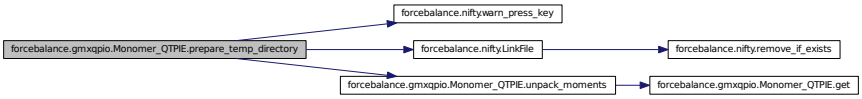
Definition at line 176 of file gmxqpio.py.

Here is the call graph for this function:

#### 8.38.3.3 def forcebalance.gmxqpio.Monomer_QTPIE.prepare_temp_directory ( *self, options, tgt_opts* )

Definition at line 197 of file gmxqpio.py.

Here is the call graph for this function:




#### 8.38.3.4 def forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments ( *self, moment_dict* )

Definition at line 214 of file gmxqpio.py.

Here is the call graph for this function:


### 8.38.4 Member Data Documentation

#### 8.38.4.1 forcebalance::gmxqpio.Monomer_QTPIE::calc_moments

Definition at line 218 of file gmxqpio.py.

#### 8.38.4.2 forcebalance::gmxqpio.Monomer_QTPIE::objective

Definition at line 218 of file gmxqpio.py.

#### 8.38.4.3 forcebalance::gmxqpio.Monomer_QTPIE::ref_moments

Definition at line 129 of file gmxqpio.py.

#### 8.38.4.4 forcebalance::gmxqpio.Monomer_QTPIE::weights

Definition at line 129 of file gmxqpio.py.

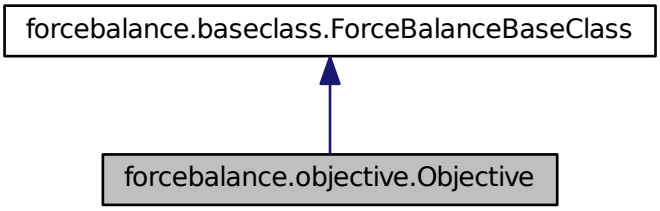
The documentation for this class was generated from the following file:

- gmxqpio.py

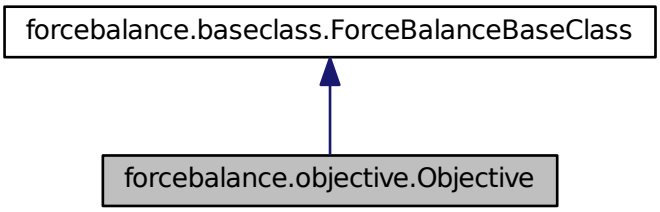
## 8.39 forcebalance.objective.Objective Class Reference

Objective function.

Inheritance diagram for forcebalance.objective.Objective:




Collaboration diagram for forcebalance.objective.Objective:




#### Public Member Functions

- def __init__
- def Target_Terms
- def Indicate

  *Print objective function contributions.*
- def Full

#### Public Attributes

- Targets

  *Work Queue Port (The specific target itself may or may not actually use this.)*
- FF

  *The force field (it seems to be everywhere)*
- Penalty

  *Initialize the penalty function.*
- WTot

*Obtain the denominator.*

- ObjDict
- ObjDict_Last

### 8.39.1 Detailed Description

Objective function.

The objective function is a combination of contributions from the different fitting targets. Basically, it loops through the targets, gets their contributions to the objective function and then sums all of them (although more elaborate schemes are conceivable). The return value is the same data type as calling the target itself: a dictionary containing the objective function, the gradient and the Hessian.

The penalty function is also computed here; it keeps the parameters from straying too far from their initial values.

**Parameters**

| | | |
|---|---|---|
| in | *mvals* | The mathematical parameters that enter into computing the objective function |
| in | *Order* | The requested order of differentiation |

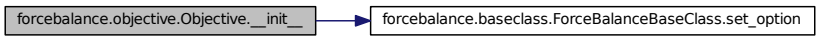
Definition at line 106 of file objective.py.

### 8.39.2 Constructor & Destructor Documentation

#### 8.39.2.1 def forcebalance.objective.Objective.__init__ ( *self, options, tgt_opts, forcefield* )

Definition at line 107 of file objective.py.

Here is the call graph for this function:




### 8.39.3 Member Function Documentation

#### 8.39.3.1 def forcebalance.objective.Objective.Full ( *self, mvals, Order* = 0, *verbose* = False )

Definition at line 250 of file objective.py.

Here is the call graph for this function:

#### 8.39.3.2 def forcebalance.objective.Objective.Indicate ( *self* )

Print objective function contributions.

Definition at line 213 of file objective.py.

Here is the call graph for this function:

#### 8.39.3.3 def forcebalance.objective.Objective.Target_Terms ( *self, mvals, Order* = 0, *verbose* = False )

Definition at line 153 of file objective.py.

Here is the call graph for this function:

### 8.39.4 Member Data Documentation

#### 8.39.4.1 forcebalance::objective.Objective::FF

The force field (it seems to be everywhere)

Definition at line 111 of file objective.py.

#### 8.39.4.2 forcebalance::objective.Objective::ObjDict

Definition at line 113 of file objective.py.

#### 8.39.4.3 forcebalance::objective.Objective::ObjDict_Last

Definition at line 113 of file objective.py.

#### 8.39.4.4 forcebalance::objective.Objective::Penalty

Initialize the penalty function.

Definition at line 112 of file objective.py.

#### 8.39.4.5 forcebalance::objective.Objective::Targets

Work Queue Port (The specific target itself may or may not actually use this.)

Asynchronous objective function evaluation (i.e. execute Work Queue and local objective concurrently.) The list of fitting targets

Definition at line 110 of file objective.py.

#### 8.39.4.6 forcebalance::objective.Objective::WTot

Obtain the denominator.

Definition at line 113 of file objective.py.

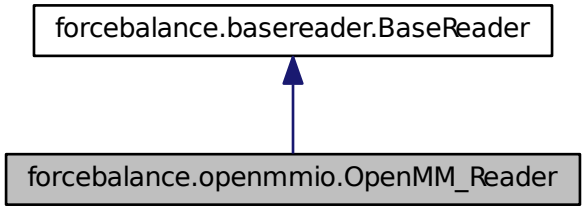
The documentation for this class was generated from the following file:

- objective.py

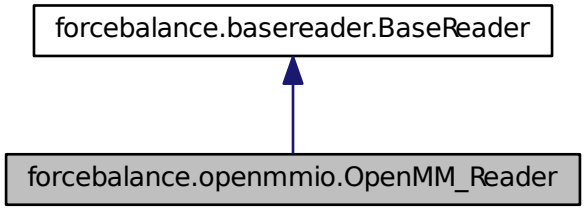
## 8.40 forcebalance.openmmio.OpenMM_Reader Class Reference

Class for parsing OpenMM force field files.

Inheritance diagram for forcebalance.openmmio.OpenMM_Reader:




Collaboration diagram for forcebalance.openmmio.OpenMM_Reader:




### Public Member Functions

- def __init__
- def build_pid

  *Build the parameter identifier (see _link_ for an example)*

### Public Attributes

- pdict

  *Initialize the superclass.*

### 8.40.1 Detailed Description

Class for parsing OpenMM force field files.

Definition at line 302 of file openmmio.py.

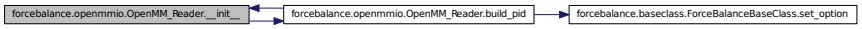
### 8.40.2 Constructor & Destructor Documentation

#### 8.40.2.1 def forcebalance.openmmio.OpenMM_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 303 of file openmmio.py.

Here is the call graph for this function:




### 8.40.3 Member Function Documentation

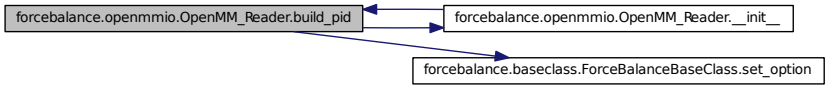
#### 8.40.3.1 def forcebalance.openmmio.OpenMM_Reader.build_pid ( *self, element, parameter* )

Build the parameter identifier (see _link_ for an example)

**Todo** Add a link here

Definition at line 312 of file openmmio.py.

Here is the call graph for this function:




### 8.40.4 Member Data Documentation

#### 8.40.4.1 forcebalance::openmmio.OpenMM_Reader::pdict

Initialize the superclass.

:) The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 305 of file openmmio.py.

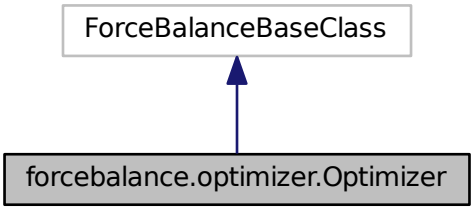
The documentation for this class was generated from the following file:

- openmmio.py

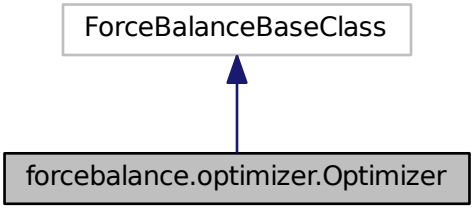
## 8.41 forcebalance.optimizer.Optimizer Class Reference

Optimizer class.

Inheritance diagram for forcebalance.optimizer.Optimizer:




Collaboration diagram for forcebalance.optimizer.Optimizer:




### Public Member Functions

- def __init__

  *Create an Optimizer object.*
- def Run

  *Call the appropriate optimizer.*
- def MainOptimizer

  *The main ForceBalance adaptive trust-radius pseudo-Newton optimizer.*
- def step

  *Computes the next step in the parameter space.*
- def NewtonRaphson

  *Optimize the force field parameters using the Newton-Raphson method (.*
- def BFGS

  *Optimize the force field parameters using the BFGS method; currently the recommended choice (.*

- def ScipyOptimizer

  *Driver for SciPy optimizations.*
- def GeneticAlgorithm

  *Genetic algorithm, under development.*
- def Simplex

  *Use SciPy's built-in simplex algorithm to optimize the parameters.*
- def Powell

  *Use SciPy's built-in Powell direction-set algorithm to optimize the parameters.*
- def Anneal

  *Use SciPy's built-in simulated annealing algorithm to optimize the parameters.*
- def ConjugateGradient

  *Use SciPy's built-in simulated annealing algorithm to optimize the parameters.*
- def Scan_Values

  *Scan through parameter values.*
- def ScanMVals

  *Scan through the mathematical parameter space.*
- def ScanPVals

  *Scan through the physical parameter space.*
- def SinglePoint

  *A single-point objective function computation.*
- def Gradient

  *A single-point gradient computation.*
- def Hessian

  *A single-point Hessian computation.*
- def FDCheckG

  *Finite-difference checker for the objective function gradient.*
- def FDCheckH

  *Finite-difference checker for the objective function Hessian.*
- def readchk

  *Read the checkpoint file for the main optimizer.*
- def writechk

  *Write the checkpoint file for the main optimizer.*

#### Public Attributes

- OptTab

  *A list of all the things we can ask the optimizer to do.*
- Objective

  *The root directory.*
- bhyp

  *Whether the penalty function is hyperbolic.*
- FF

  *The force field itself.*
- excision

  *The indices to be excluded from the Hessian update.*
- np

  *Number of parameters.*

- mvals0

  *The original parameter values.*
- chk

  *Put data into the checkpoint file.*
- H
- dx
- Val
- Grad
- Hess
- Penalty

#### 8.41.1 Detailed Description

Optimizer class.

Contains several methods for numerical optimization.

For various reasons, the optimizer depends on the force field and fitting targets (i.e. we cannot treat it as a fully independent numerical optimizer). The dependency is rather weak which suggests that I can remove it someday.

Definition at line 42 of file optimizer.py.

#### 8.41.2 Constructor & Destructor Documentation

##### 8.41.2.1 def forcebalance.optimizer.Optimizer.__init__ ( *self, options, Objective, FF* )

Create an Optimizer object.

The optimizer depends on both the FF and the fitting targets so there is a chain of dependencies: FF --> FitSim --> Optimizer, and FF --> Optimizer

Here's what we do:

- Take options from the parser
- Pass in the objective function, force field, all fitting targets

Definition at line 55 of file optimizer.py.

#### 8.41.3 Member Function Documentation

##### 8.41.3.1 def forcebalance.optimizer.Optimizer.Anneal ( *self* )

Use SciPy's built-in simulated annealing algorithm to optimize the parameters.

**See also**

Optimizer::ScipyOptimizer

Definition at line 780 of file optimizer.py.

#### 8.41.3.2 def forcebalance.optimizer.Optimizer.BFGS ( *self* )

Optimize the force field parameters using the BFGS method; currently the recommended choice (.

**See also**

MainOptimizer)

Definition at line 600 of file optimizer.py.

#### 8.41.3.3 def forcebalance.optimizer.Optimizer.ConjugateGradient ( *self* )

Use SciPy's built-in simulated annealing algorithm to optimize the parameters.

**See also**

Optimizer::ScipyOptimizer

Definition at line 785 of file optimizer.py.

#### 8.41.3.4 def forcebalance.optimizer.Optimizer.FDCheckG ( *self* )

Finite-difference checker for the objective function gradient.

For each element in the gradient, use a five-point finite difference stencil to compute a finite-difference derivative, and compare it to the analytic result.

Definition at line 881 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.5 def forcebalance.optimizer.Optimizer.FDCheckH ( *self* )

Finite-difference checker for the objective function Hessian.

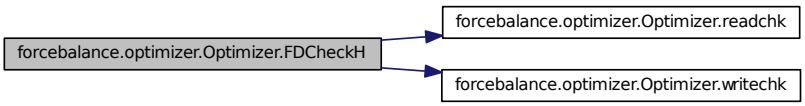
For each element in the Hessian, use a five-point stencil in both parameter indices to compute a finite-difference derivative, and compare it to the analytic result.

This is meant to be a foolproof checker, so it is pretty slow. We could write a faster checker if we assumed we had accurate first derivatives, but it's better to not make that assumption.

The second derivative is computed by double-wrapping the objective function via the 'wrap2' function.

Definition at line 913 of file optimizer.py.

Here is the call graph for this function:




#### 8.41.3.6 def forcebalance.optimizer.Optimizer.GeneticAlgorithm ( *self* )

Genetic algorithm, under development.

It currently works but a genetic algorithm is more like a concept; i.e. there is no single way to implement it.

**Todo** Massive parallelization hasn't been implemented yet

Definition at line 677 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.7 def forcebalance.optimizer.Optimizer.Gradient ( *self* )

A single-point gradient computation.

Definition at line 859 of file optimizer.py.

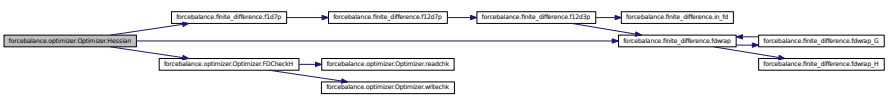
Here is the call graph for this function:

#### 8.41.3.8 def forcebalance.optimizer.Optimizer.Hessian ( *self* )

A single-point Hessian computation.

Definition at line 867 of file optimizer.py.

Here is the call graph for this function:



#### 8.41.3.9 def forcebalance.optimizer.Optimizer.MainOptimizer ( *self, b_BFGS* = 0 )

The main ForceBalance adaptive trust-radius pseudo-Newton optimizer.

Tried and true in many situations. :)

Usually this function is called with the BFGS or NewtonRaphson method. The NewtonRaphson method is consistently the best method I have, because I always provide at least an approximate Hessian to the objective function. The BFGS method is vestigial and currently does not work.

BFGS is a pseudo-Newton method in the sense that it builds an approximate Hessian matrix from the gradient information in previous steps; Newton-Raphson requires the actual Hessian matrix. However, the algorithms are similar in that they both compute the step by inverting the Hessian and multiplying by the gradient.

The method adaptively changes the step size. If the step is sufficiently good (i.e. the objective function goes down by a large fraction of the predicted decrease), then the step size is increased; if the step is bad, then it rejects the step and tries again.

The optimization is terminated after either a function value or step size tolerance is reached.

**Parameters**

| | | |
|---|---|---|
| in | *b_BFGS* | Switch to use BFGS (True) or Newton-Raphson (False) |

Definition at line 228 of file optimizer.py.

#### 8.41.3.10 def forcebalance.optimizer.Optimizer.NewtonRaphson ( *self* )

Optimize the force field parameters using the Newton-Raphson method (.

**See also**

MainOptimizer)

Definition at line 595 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.11 def forcebalance.optimizer.Optimizer.Powell ( *self* )

Use SciPy's built-in Powell direction-set algorithm to optimize the parameters.

**See also**

Optimizer::ScipyOptimizer

Definition at line 775 of file optimizer.py.

#### 8.41.3.12 def forcebalance.optimizer.Optimizer.readchk ( *self* )

Read the checkpoint file for the main optimizer.

Definition at line 941 of file optimizer.py.

#### 8.41.3.13 def forcebalance.optimizer.Optimizer.Run ( *self* )

Call the appropriate optimizer.

This is the method we might want to call from an executable.

Definition at line 166 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.14 def forcebalance.optimizer.Optimizer.Scan_Values ( *self, MathPhys* = `1` )

Scan through parameter values.

This option is activated using the inputs:

```
scan[mp]vals
scan_vals low:hi:nsteps
scan_idxnum (number) -or-
scan_idxname (name)
```

This method goes to the specified parameter indices and scans through the supplied values, evaluating the objective function at every step.

I hope this method will be useful for people who just want to look at changing one or two parameters and seeing how it affects the force field performance.

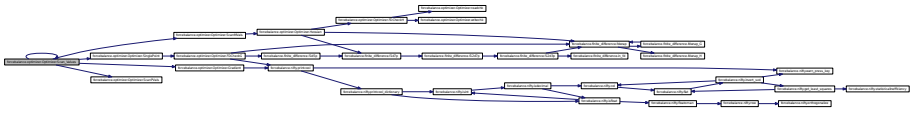
**Todo** Maybe a multidimensional grid can be done.

**Parameters**

| | | |
|---|---|---|
| `in` | *MathPhys* | Switch to use mathematical (True) or physical (False) parameters. |

Definition at line 811 of file optimizer.py.

Here is the call graph for this function:



#### 8.41.3.15 def forcebalance.optimizer.Optimizer.ScanMVals ( *self* )

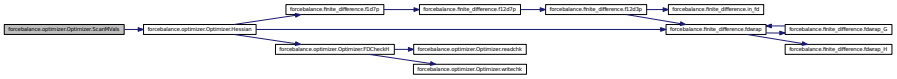
Scan through the mathematical parameter space.

**See also**

Optimizer::ScanValues

Definition at line 843 of file optimizer.py.

Here is the call graph for this function:



#### 8.41.3.16 def forcebalance.optimizer.Optimizer.ScanPVals ( *self* )

Scan through the physical parameter space.

**See also**

Optimizer::ScanValues

Definition at line 848 of file optimizer.py.

#### 8.41.3.17 def forcebalance.optimizer.Optimizer.ScipyOptimizer ( *self, Algorithm* = "None" )

Driver for SciPy optimizations.

Using any of the SciPy optimizers requires that SciPy is installed. This method first defines several wrappers around the objective function that the SciPy optimizers can use. Then it calls the algorith mitself.

**Parameters**

| | | |
|---|---|---|
| in | *Algorithm* | The optimization algorithm to use, for example 'powell', 'simplex' or 'anneal' |

Definition at line 613 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.18 def forcebalance.optimizer.Optimizer.Simplex ( *self* )

Use SciPy's built-in simplex algorithm to optimize the parameters.

**See also**

Optimizer::ScipyOptimizer

Definition at line 770 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.19 def forcebalance.optimizer.Optimizer.SinglePoint ( *self* )

A single-point objective function computation.

Definition at line 853 of file optimizer.py.

Here is the call graph for this function:

#### 8.41.3.20 def forcebalance.optimizer.Optimizer.step ( *self, xk, data, trust* )

Computes the next step in the parameter space.

There are lots of tricks here that I will document later.

**Parameters**

| | | |
|---|---|---|
| in | *G* | The gradient |
| in | *H* | The Hessian |
| in | *trust* | The trust radius |

Definition at line 420 of file optimizer.py.

#### 8.41.3.21 def forcebalance.optimizer.Optimizer.writechk ( *self* )

Write the checkpoint file for the main optimizer.

Definition at line 953 of file optimizer.py.

### 8.41.4 Member Data Documentation

#### 8.41.4.1 forcebalance::optimizer.Optimizer::bhyp

Whether the penalty function is hyperbolic.

Definition at line 83 of file optimizer.py.

#### 8.41.4.2 forcebalance::optimizer.Optimizer::chk

Put data into the checkpoint file.

Definition at line 229 of file optimizer.py.

#### 8.41.4.3 forcebalance::optimizer.Optimizer::dx

Definition at line 422 of file optimizer.py.

#### 8.41.4.4 forcebalance::optimizer.Optimizer::excision

The indices to be excluded from the Hessian update.

Definition at line 85 of file optimizer.py.

#### 8.41.4.5 forcebalance::optimizer.Optimizer::FF

The force field itself.

Definition at line 84 of file optimizer.py.

#### 8.41.4.6 forcebalance::optimizer.Optimizer::Grad

Definition at line 422 of file optimizer.py.

#### 8.41.4.7 forcebalance::optimizer.Optimizer::H

Definition at line 422 of file optimizer.py.

#### 8.41.4.8 forcebalance::optimizer.Optimizer::Hess

Definition at line 422 of file optimizer.py.

#### 8.41.4.9 forcebalance::optimizer.Optimizer::mvals0

The original parameter values.

Sometimes the optimizer doesn't return anything (i.e.

in the case of a single point calculation) In these situations, don't do anything Check derivatives by finite difference after the optimization is over (for good measure)

Definition at line 87 of file optimizer.py.

#### 8.41.4.10 forcebalance::optimizer.Optimizer::np

Number of parameters.

Definition at line 86 of file optimizer.py.

#### 8.41.4.11 forcebalance::optimizer.Optimizer::Objective

The root directory.

The job type Initial step size trust radius Minimum trust radius (for noisy objective functions) Lower bound on Hessian eigenvalue (below this, we add in steepest descent) Guess value for Brent Step size for numerical finite difference Number of steps to average over Function value convergence threshold Step size convergence threshold Gradient convergence threshold Maximum number of optimization steps For scan[mp]vals: The parameter index to scan over For scan[mp]vals: The parameter name to scan over, it just looks up an index For scan[mp]vals: The values that are fed into the scanner Name of the checkpoint file that we're reading in Name of the checkpoint file that we're writing

out Whether to write the checkpoint file at every step Adaptive trust radius adjustment factor Adaptive trust radius adjustment damping Whether to print gradient during each step of the optimization Whether to print Hessian during each step of the optimization Whether to print parameters during each step of the optimization Error tolerance (if objective function rises by less than this, then the optimizer will forge ahead!) Search tolerance (The nonlinear search will stop if the change is below this threshold) The objective function (needs to pass in when I instantiate)

Definition at line 82 of file optimizer.py.

#### 8.41.4.12 forcebalance::optimizer.Optimizer::OptTab

A list of all the things we can ask the optimizer to do.

Definition at line 56 of file optimizer.py.

#### 8.41.4.13 forcebalance::optimizer.Optimizer::Penalty

Definition at line 422 of file optimizer.py.

#### 8.41.4.14 forcebalance::optimizer.Optimizer::Val

Definition at line 422 of file optimizer.py.

The documentation for this class was generated from the following file:

- optimizer.py

## 8.42 forcebalance.objective.Penalty Class Reference

Penalty functions for regularizing the force field optimizer.

### Public Member Functions

- def __init__
- def compute
- def L2_norm

  *Harmonic L2-norm constraints.*
- def HYP

  *Hyperbolic constraints.*
- def FUSE
- def FUSE_BARRIER
- def FUSE_L0

### Public Attributes

- fadd
- fmul
- a
- b
- FF
- ptyp
- Pen_Tab
- spacings

  *Find exponential spacings.*

**Static Public Attributes**

- dictionary Pen_Names

### 8.42.1 Detailed Description

Penalty functions for regularizing the force field optimizer.

The purpose for this module is to improve the behavior of our optimizer; essentially, our problem is fraught with 'linear dependencies', a.k.a. directions in the parameter space that the objective function does not respond to. This would happen if a parameter is just plain useless, or if there are two or more parameters that describe the same thing.

To accomplish these objectives, a penalty function is added to the objective function. Generally, the more the parameters change (i.e. the greater the norm of the parameter vector), the greater the penalty. Note that this is added on after all of the other contributions have been computed. This only matters if the penalty 'multiplies' the objective function: Obj + Obj∗Penalty, but we also have the option of an additive penalty: Obj + Penalty.

Statistically, this is called regularization. If the penalty function is the norm squared of the parameter vector, it is called ridge regression. There is also the option of using simply the norm, and this is called lasso, but I think it presents problems for the optimizer that I need to work out.

Note that the penalty functions can be considered as part of a 'maximum likelihood' framework in which we assume a PRIOR PROBABILITY of the force field parameters around their initial values. The penalty function is related to the prior by an exponential. Ridge regression corresponds to a Gaussian prior and lasso corresponds to an exponential prior. There is also 'elastic net regression' which interpolates between Gaussian and exponential using a tuning parameter.

Our priors are adjustable too - there is one parameter, which is the width of the distribution. We can even use a noninformative prior for the distribution widths (hyperprior!). These are all important things to consider later.

Importantly, note that here there is no code that treats the distribution width. That is because the distribution width is wrapped up in the rescaling factors, which is essentially a coordinate transformation on the parameter space. More documentation on this will follow, perhaps in the 'rsmake' method.

Definition at line 307 of file objective.py.

### 8.42.2 Constructor & Destructor Documentation

#### 8.42.2.1 def forcebalance.objective.Penalty.__init__ ( *self, User_Option, ForceField, Factor_Add* = `0.0`, *Factor_Mult* = `0.0`, *Factor_B* = `0.1`, *Alpha* = `1.0` )

Definition at line 313 of file objective.py.

### 8.42.3 Member Function Documentation

#### 8.42.3.1 def forcebalance.objective.Penalty.compute ( *self, mvals, Objective* )

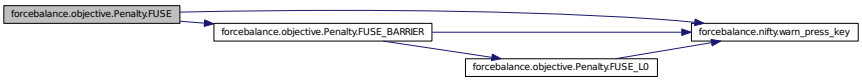
Definition at line 339 of file objective.py.

Here is the call graph for this function:

#### 8.42.3.2 def forcebalance.objective.Penalty.FUSE ( *self, mvals* )

Definition at line 399 of file objective.py.

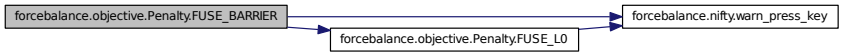
Here is the call graph for this function:




#### 8.42.3.3 def forcebalance.objective.Penalty.FUSE_BARRIER ( *self, mvals* )

Definition at line 440 of file objective.py.

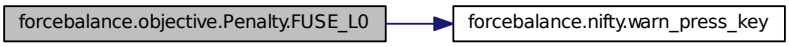
Here is the call graph for this function:




#### 8.42.3.4 def forcebalance.objective.Penalty.FUSE_L0 ( *self, mvals* )

Definition at line 482 of file objective.py.

Here is the call graph for this function:




#### 8.42.3.5 def forcebalance.objective.Penalty.HYP ( *self, mvals* )

Hyperbolic constraints.

Depending on the 'b' parameter, the smaller it is, the closer we are to an L1-norm constraint. If we use these, we expect a properly-behaving optimizer to make several of the parameters very nearly zero (which would be cool).

**Parameters**

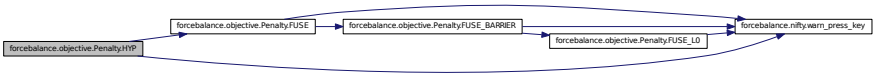
| | | |
|---|---|---|
| in | *mvals* | The parameter vector |

**Returns**

DC0 The hyperbolic penalty
DC1 The gradient
DC2 The Hessian

Definition at line 391 of file objective.py.

Here is the call graph for this function:




#### 8.42.3.6 def forcebalance.objective.Penalty.L2_norm ( *self, mvals* )

Harmonic L2-norm constraints.

These are the ones that I use the most often to regularize my optimization.

**Parameters**

| | | |
|---|---|---|
| in | *mvals* | The parameter vector |

**Returns**

DC0 The norm squared of the vector
DC1 The gradient of DC0
DC2 The Hessian (just a constant)

Definition at line 371 of file objective.py.

Here is the call graph for this function:


### 8.42.4 Member Data Documentation

#### 8.42.4.1 forcebalance::objective.Penalty::a

Definition at line 313 of file objective.py.

#### 8.42.4.2 forcebalance::objective.Penalty::b

Definition at line 313 of file objective.py.

#### 8.42.4.3 forcebalance::objective.Penalty::fadd

Definition at line 313 of file objective.py.

#### 8.42.4.4 forcebalance::objective.Penalty::FF

Definition at line 313 of file objective.py.

#### 8.42.4.5 forcebalance::objective.Penalty::fmul

Definition at line 313 of file objective.py.

#### 8.42.4.6 dictionary forcebalance::objective.Penalty::Pen_Names [static]

**Initial value:**

```
{'HYP' : 1, 'HYPER' : 1, 'HYPERBOLIC' : 1, 'L1' : 1, 'HYPERBOLA' : 1,
                    'PARA' : 2, 'PARABOLA' : 2, 'PARABOLIC' : 2, 'L2': 2, '
      QUADRATIC' : 2,
                    'FUSE' : 3, 'FUSION' : 3, 'FUSE_L0' : 4, 'FUSION_L0' : 4,
       'FUSION-L0' : 4,
                    'FUSE-BARRIER' : 5, 'FUSE-BARRIER' : 5, 'FUSE_BARRIER' :
      5, 'FUSION_BARRIER' : 5}
```

Definition at line 308 of file objective.py.

#### 8.42.4.7 forcebalance::objective.Penalty::Pen_Tab

Definition at line 313 of file objective.py.

#### 8.42.4.8 forcebalance::objective.Penalty::ptyp

Definition at line 313 of file objective.py.

#### 8.42.4.9 forcebalance::objective.Penalty::spacings

Find exponential spacings.

Definition at line 314 of file objective.py.

The documentation for this class was generated from the following file:

- objective.py

## 8.43 forcebalance.nifty.Pickler_LP Class Reference

A subclass of the python Pickler that implements pickling of _ElementTree types.

### Public Member Functions

- def __init__

### 8.43.1 Detailed Description

A subclass of the python Pickler that implements pickling of _ElementTree types.

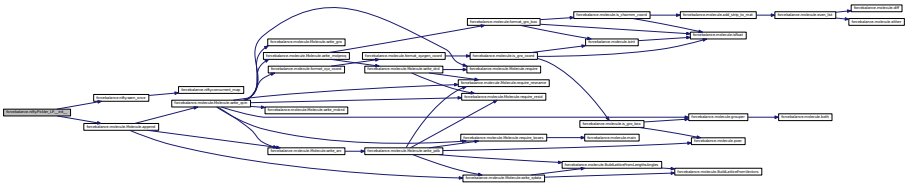
Definition at line 496 of file nifty.py.

### 8.43.2 Constructor & Destructor Documentation

#### 8.43.2.1 def forcebalance.nifty.Pickler_LP.__init__ ( *self, file, protocol* = None )

Definition at line 497 of file nifty.py.

Here is the call graph for this function:



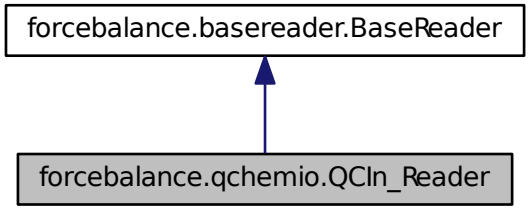
The documentation for this class was generated from the following file:

- nifty.py

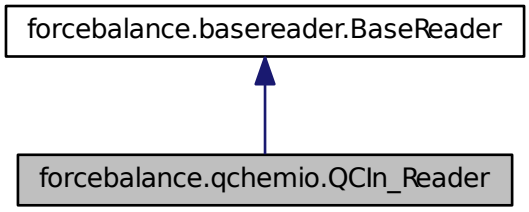
## 8.44 forcebalance.qchemio.QCIn_Reader Class Reference

Finite state machine for parsing Q-Chem input files.

Inheritance diagram for forcebalance.qchemio.QCIn_Reader:




Collaboration diagram for forcebalance.qchemio.QCIn_Reader:



**Public Member Functions**

- def __init__
- def feed

  *Feed in a line.*

**Public Attributes**

- atom
- snum
- cnum
- shell
- pdict
- sec
- itype
- suffix

### 8.44.1 Detailed Description

Finite state machine for parsing Q-Chem input files.

Definition at line 28 of file qchemio.py.

### 8.44.2 Constructor & Destructor Documentation

#### 8.44.2.1 def forcebalance.qchemio.QCIn_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 30 of file qchemio.py.

### 8.44.3 Member Function Documentation

#### 8.44.3.1 def forcebalance.qchemio.QCIn_Reader.feed ( *self, line* )

Feed in a line.

**Parameters**

| | | |
|---|---|---|
| in | *line* | The line of data |

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 45 of file qchemio.py.

### 8.44.4 Member Data Documentation

#### 8.44.4.1 forcebalance::qchemio.QCIn_Reader::atom

Definition at line 30 of file qchemio.py.

#### 8.44.4.2 forcebalance::qchemio.QCIn_Reader::cnum

Definition at line 30 of file qchemio.py.

#### 8.44.4.3 forcebalance::qchemio.QCIn_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 45 of file qchemio.py.

#### 8.44.4.4 forcebalance::qchemio.QCIn_Reader::pdict

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 30 of file qchemio.py.

#### 8.44.4.5 forcebalance::qchemio.QCIn_Reader::sec

Definition at line 45 of file qchemio.py.

#### 8.44.4.6 forcebalance::qchemio.QCIn_Reader::shell

Definition at line 30 of file qchemio.py.

#### 8.44.4.7 forcebalance::qchemio.QCIn_Reader::snum

Definition at line 30 of file qchemio.py.

#### 8.44.4.8 forcebalance::qchemio.QCIn_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 45 of file qchemio.py.

The documentation for this class was generated from the following file:

- qchemio.py

## 8.45 forcebalance.nifty.RawFileHandler Class Reference

Exactly like logging.FileHandler except it does no extra formatting before sending logging messages to the file.

### Public Member Functions

- def emit

### 8.45.1 Detailed Description

Exactly like logging.FileHandler except it does no extra formatting before sending logging messages to the file.

This is more compatible with how output has been displayed in ForceBalance.

Definition at line 925 of file nifty.py.

### 8.45.2 Member Function Documentation

#### 8.45.2.1 def forcebalance.nifty.RawFileHandler.emit ( *self, record* )

Definition at line 926 of file nifty.py.

Here is the call graph for this function:


The documentation for this class was generated from the following file:

- nifty.py

## 8.46 forcebalance.nifty.RawStreamHandler Class Reference

Exactly like logging.StreamHandler except it does no extra formatting before sending logging messages to the stream.

### Public Member Functions

- def __init__
- def emit

### 8.46.1 Detailed Description

Exactly like logging.StreamHandler except it does no extra formatting before sending logging messages to the stream.

This is more compatible with how output has been displayed in ForceBalance. Default stream has also been changed from stderr to stdout

Definition at line 912 of file nifty.py.

### 8.46.2 Constructor & Destructor Documentation

#### 8.46.2.1 def forcebalance.nifty.RawStreamHandler.__init__ ( *self, stream* = `sys.stdout` )

Definition at line 913 of file nifty.py.

### 8.46.3 Member Function Documentation

#### 8.46.3.1 def forcebalance.nifty.RawStreamHandler.emit ( *self, record* )

Definition at line 916 of file nifty.py.

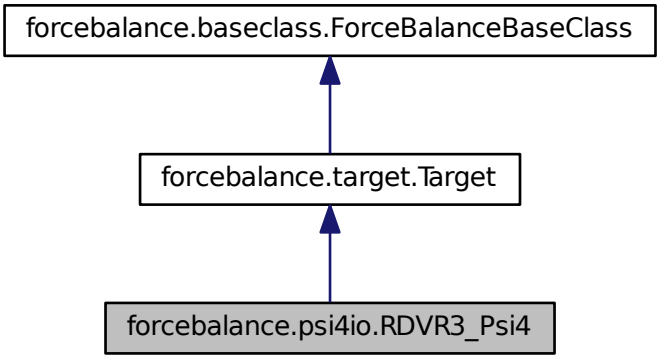
The documentation for this class was generated from the following file:

- nifty.py

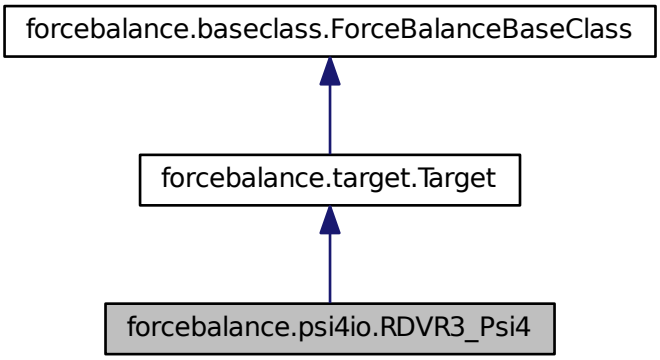
## 8.47 forcebalance.psi4io.RDVR3_Psi4 Class Reference

Subclass of Target for R-DVR3 grid fitting.

Inheritance diagram for forcebalance.psi4io.RDVR3_Psi4:




Collaboration diagram for forcebalance.psi4io.RDVR3_Psi4:




### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def indicate

- def driver
- def get

  *LPW 04-17-2013.*

#### Public Attributes

- objfiles

  *Which parameters are differentiated?*
- objvals
- elements
- molecules
- callderivs
- factor
- tdir
- objd
- gradd
- hdiagd
- objective

### 8.47.1 Detailed Description

Subclass of Target for R-DVR3 grid fitting.

Main features:

- Multiple molecules are treated as a single target.
- R-DVR3 can only print out the objective function, it cannot print out the residual vector.
- We should be smart enough to mask derivatives.

Definition at line 295 of file psi4io.py.

### 8.47.2 Constructor & Destructor Documentation

#### 8.47.2.1 def forcebalance.psi4io.RDVR3_Psi4.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.target.Target.

Definition at line 298 of file psi4io.py.

### 8.47.3 Member Function Documentation

#### 8.47.3.1 def forcebalance.psi4io.RDVR3_Psi4.driver ( *self, mvals, d* )

Definition at line 346 of file psi4io.py.

Here is the call graph for this function:


#### 8.47.3.2 def forcebalance.psi4io.RDVR3_Psi4.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

LPW 04-17-2013.

This subroutine builds the objective function from Psi4.

**Parameters**

| | | |
|---|---|---|
| `in` | *mvals* | Mathematical parameter values |
| `in` | *AGrad* | Switch to turn on analytic gradient |
| `in` | *AHess* | Switch to turn on analytic Hessian |

**Returns**

Answer Contribution to the objective function

Reimplemented from forcebalance.target.Target.

Definition at line 382 of file psi4io.py.

#### 8.47.3.3 def forcebalance.psi4io.RDVR3_Psi4.indicate ( *self* )

Definition at line 337 of file psi4io.py.

Here is the call graph for this function:


### 8.47.4 Member Data Documentation

#### 8.47.4.1 forcebalance::psi4io.RDVR3_Psi4::callderivs

Definition at line 299 of file psi4io.py.

#### 8.47.4.2 forcebalance::psi4io.RDVR3_Psi4::elements

Definition at line 299 of file psi4io.py.

#### 8.47.4.3 forcebalance::psi4io.RDVR3_Psi4::factor

Definition at line 299 of file psi4io.py.

#### 8.47.4.4 forcebalance::psi4io.RDVR3_Psi4::gradd

Definition at line 382 of file psi4io.py.

#### 8.47.4.5 forcebalance::psi4io.RDVR3_Psi4::hdiagd

Definition at line 382 of file psi4io.py.

#### 8.47.4.6 forcebalance::psi4io.RDVR3_Psi4::molecules

Definition at line 299 of file psi4io.py.

#### 8.47.4.7 forcebalance::psi4io.RDVR3_Psi4::objd

Definition at line 382 of file psi4io.py.

#### 8.47.4.8 forcebalance::psi4io.RDVR3_Psi4::objective

Definition at line 382 of file psi4io.py.

#### 8.47.4.9 forcebalance::psi4io.RDVR3_Psi4::objfiles

Which parameters are differentiated?

Definition at line 299 of file psi4io.py.

#### 8.47.4.10 forcebalance::psi4io.RDVR3_Psi4::objvals

Definition at line 299 of file psi4io.py.

#### 8.47.4.11 forcebalance::psi4io.RDVR3_Psi4::tdir

Definition at line 382 of file psi4io.py.

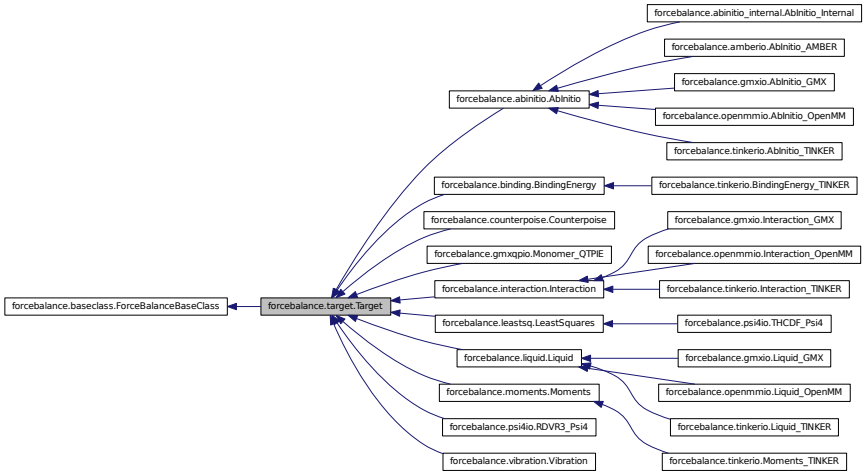
The documentation for this class was generated from the following file:

- psi4io.py

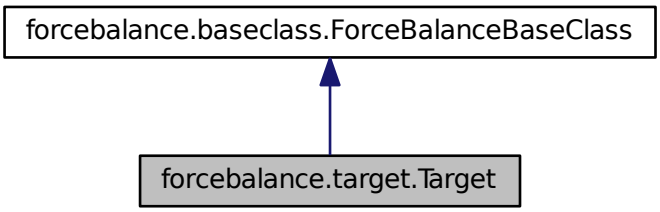
## 8.48 forcebalance.target.Target Class Reference

Base class for all fitting targets.

Inheritance diagram for forcebalance.target.Target:




Collaboration diagram for forcebalance.target.Target:




### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def get_X

  *Computes the objective function contribution without any parametric derivatives.*
- def get_G

  *Computes the objective function contribution and its gradient.*
- def get_H

  *Computes the objective function contribution and its gradient / Hessian.*
- def link_from_tempdir
- def refresh_temp_directory

  *Back up the temporary directory if desired, delete it and then create a new one.*

- def get

  *Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.*
- def sget

  *Stages the directory for the target, and then calls 'get'.*
- def submit_jobs
- def stage

  *Stages the directory for the target, and then launches Work Queue processes if any.*
- def wq_complete

  *This method determines whether the Work Queue tasks for the current target have completed.*
- def printcool_table

  *Print target information in an organized table format.*

#### Public Attributes

- tempdir

  *Root directory of the whole project.*
- rundir

  *The directory in which the simulation is running - this can be updated.*
- FF

  *Need the forcefield (here for now)*
- xct

  *Counts how often the objective function was computed.*
- gct

  *Counts how often the gradient was computed.*
- hct

  *Counts how often the Hessian was computed.*

### 8.48.1 Detailed Description

Base class for all fitting targets.

In ForceBalance a Target is defined as a set of reference data plus a corresponding method to simulate that data using the force field.

The 'computable quantities' may include energies and forces where the reference values come from QM calculations (energy and force matching), energies from an EDA analysis (Maybe in the future, FDA?), molecular properties (like polarizability, refractive indices, multipole moments or vibrational frequencies), relative entropies, and bulk properties. Single-molecule or bulk properties can even come from the experiment!

The central idea in ForceBalance is that each quantity makes a contribution to the overall objective function. So we can build force fields that fit several quantities at once, rather than putting all of our chips behind energy and force matching. In the future ForceBalance may even include multiobjective optimization into the optimizer.

The optimization is done by way of minimizing an 'objective function', which is comprised of squared differences between the computed and reference values. These differences are not computed in this file, but rather in subclasses that use Target as a base class. Thus, the contents of Target itself are meant to be as general as possible, because the pertinent variables apply to all types of fitting targets.

An important node: Target requires that all subclasses have a method get(self,mvals,AGrad=False,AHess=False) that does the following:

Inputs: mvals = The parameter vector, which modifies the force field (Note to self: We include mvals with each Target because we can create copies of the force field and do finite difference derivatives) AGrad, AHess = Boolean switches for computing analytic gradients and Hessians

Outputs: Answer = {'X': Number, 'G': numpy.array(np), 'H': numpy.array((np,np)) } 'X' = The objective function itself 'G' = The gradient, elements not computed analytically are zero 'H' = The Hessian, elements not computed analytically are zero

This is the only global requirement of a Target. Obviously 'get' itself is not defined here, because its calculation will depend entirely on specifically which target we wish to use. However, this should give us a unified framework which will faciliate rapid implementation of Targets.

Future work: Robert suggested that I could enable automatic detection of which parameters need to be computed by finite difference. Not a bad idea. :)

Definition at line 72 of file target.py.

### 8.48.2 Constructor & Destructor Documentation

#### 8.48.2.1 def forcebalance.target.Target.__init__ ( *self, options, tgt_opts, forcefield* )

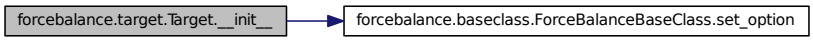
All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented in forcebalance.gmxio.Interaction_GMX, forcebalance.tinkerio.Interaction_TINKER, forcebalance.gmxio.Liquid_GMX, forcebalance.openmmio.Interaction_OpenMM, forcebalance.tinkerio.BindingEnergy_TINKER, forcebalance.gmxio.AbInitio_GMX, forcebalance.tinkerio.Moments_TINKER, forcebalance.openmmio.AbInitio_OpenMM, forcebalance.openmmio.Liquid_OpenMM, forcebalance.psi4io.RDVR3_Psi4, forcebalance.tinkerio.AbInitio_TINKER, forcebalance.tinkerio.Liquid_TINKER, forcebalance.amberio.AbInitio_AMBER, forcebalance.gmxqpio.Monomer_QTPIE, forcebalance.binding.BindingEnergy, forcebalance.psi4io.THCDF_Psi4, forcebalance.liquid.Liquid, forcebalance.abinitio.AbInitio, forcebalance.abinitio_internal.AbInitio_Internal, forcebalance.counterpoise.Counterpoise, forcebalance.interaction.Interaction, forcebalance.leastsq.LeastSquares, forcebalance.moments.Moments, and forcebalance.vibration.Vibration.

Definition at line 89 of file target.py.

Here is the call graph for this function:




### 8.48.3 Member Function Documentation

#### 8.48.3.1 def forcebalance.target.Target.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Every target must be able to return a contribution to the objective function - however, this must be implemented in the specific subclass.

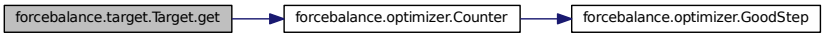
See abinitio for an example.

Reimplemented in forcebalance.abinitio.AbInitio, forcebalance.liquid.Liquid, forcebalance.psi4io.RDVR3_Psi4,

forcebalance.gmxqpio.Monomer_QTPIE, forcebalance.moments.Moments, forcebalance.binding.BindingEnergy, forcebalance.interaction.Interaction, forcebalance.counterpoise.Counterpoise, forcebalance.vibration.Vibration, and forcebalance.leastsq.LeastSquares.

Definition at line 251 of file target.py.

Here is the call graph for this function:



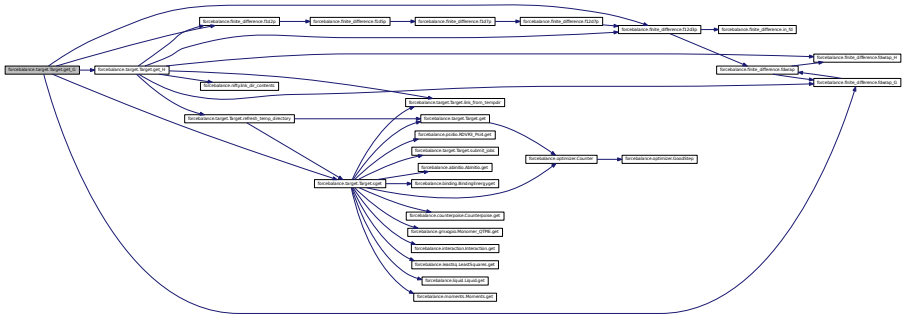

#### 8.48.3.2 def forcebalance.target.Target.get_G ( *self, mvals* = None )

Computes the objective function contribution and its gradient.

First the low-level 'get' method is called with the analytic gradient switch turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on. Alternately we can compute the gradient elements and diagonal Hessian elements at the same time using central difference if 'fdhessdiag' is turned on.

Definition at line 169 of file target.py.

Here is the call graph for this function:



#### 8.48.3.3 def forcebalance.target.Target.get_H ( *self, mvals* = None )

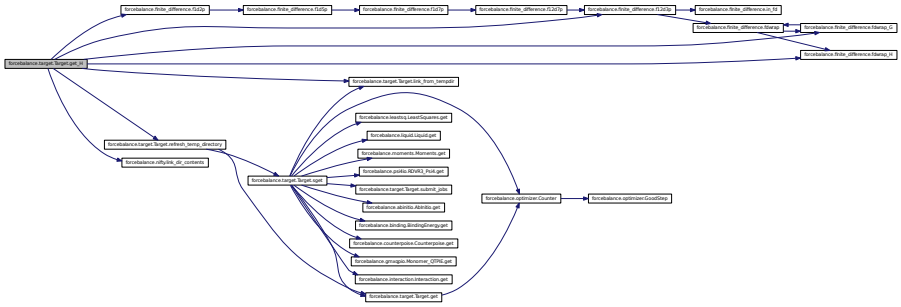
Computes the objective function contribution and its gradient / Hessian.

First the low-level 'get' method is called with the analytic gradient and Hessian both turned on. Then we loop through the fd1_pids and compute the corresponding elements of the gradient by finite difference, if the 'fdgrad' switch is turned on.

This is followed by looping through the fd2_pids and computing the corresponding Hessian elements by finite difference. Forward finite difference is used throughout for the sake of speed.

Definition at line 192 of file target.py.

Here is the call graph for this function:

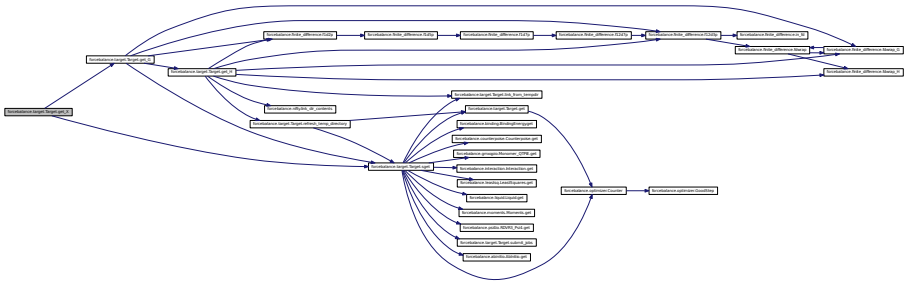


#### 8.48.3.4 def forcebalance.target.Target.get_X ( *self, mvals* = None )

Computes the objective function contribution without any parametric derivatives.

Definition at line 152 of file target.py.

Here is the call graph for this function:



#### 8.48.3.5 def forcebalance.target.Target.link_from_tempdir ( *self, absdestdir* )

Definition at line 210 of file target.py.

#### 8.48.3.6 def forcebalance.target.Target.printcool_table ( *self, data* = OrderedDict([]), *headings* = [], *banner* = None, *footnote* = None, *color* = 0 )

Print target information in an organized table format.

Implemented 6/30 because multiple targets are already printing out tabulated information in very similar ways. This method is a simple wrapper around printcool_dictionary.

The input should be something like:

**Parameters**

| | |
|---|---|
| *data* | Column contents in the form of an OrderedDict, with string keys and list vals. The key is printed in the leftmost column and the vals are printed in the other columns. If non-strings are passed, they will be converted to strings (not recommended). |
| *headings* | Column headings in the form of a list. It must be equal to the number to the list length for each of the "vals" in OrderedDict, plus one. Use "\n" characters to specify long column names that may take up more than one line. |

| | |
|---|---|
| *banner* | Optional heading line, which will be printed at the top in the title. |
| *footnote* | Optional footnote line, which will be printed at the bottom. |

Definition at line 364 of file target.py.

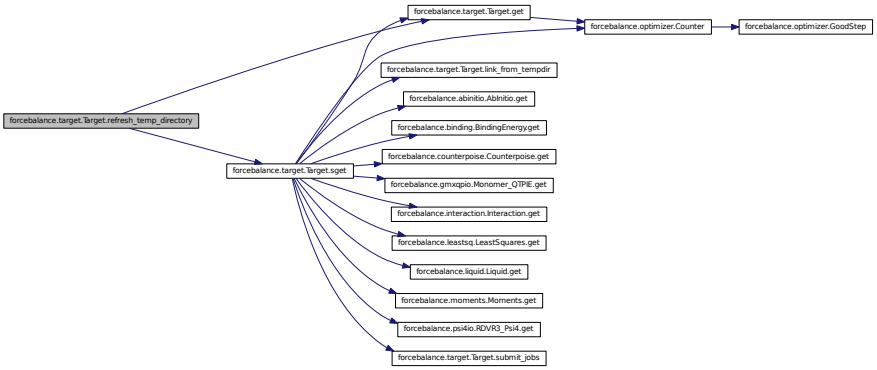
Here is the call graph for this function:


#### 8.48.3.7 def forcebalance.target.Target.refresh_temp_directory ( *self* )

Back up the temporary directory if desired, delete it and then create a new one.

Definition at line 216 of file target.py.

Here is the call graph for this function:



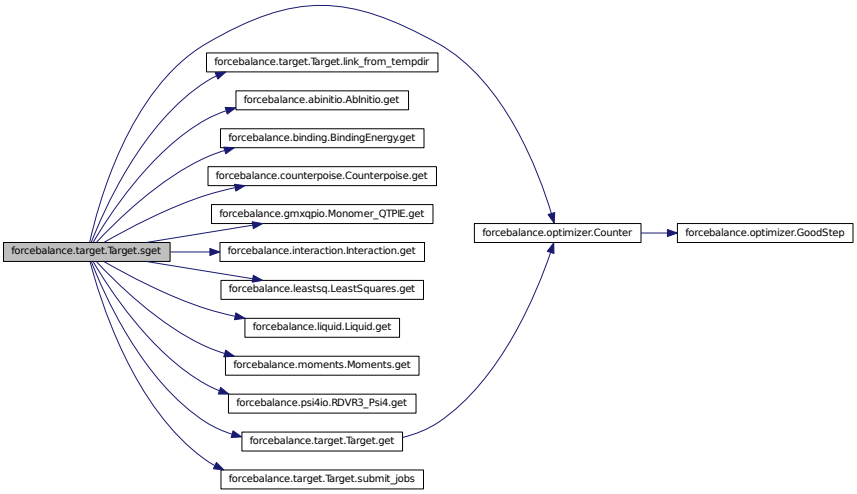

#### 8.48.3.8 def forcebalance.target.Target.sget ( *self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None` )

Stages the directory for the target, and then calls 'get'.

The 'get' method should not worry about the directory that it's running in.

Definition at line 263 of file target.py.

Here is the call graph for this function:




#### 8.48.3.9 def forcebalance.target.Target.stage ( *self, mvals, AGrad* = `False`, *AHess* = `False`, *customdir* = `None` )

Stages the directory for the target, and then launches Work Queue processes if any.

The 'get' method should not worry about the directory that it's running in.

Definition at line 298 of file target.py.

Here is the call graph for this function:

#### 8.48.3.10 def forcebalance.target.Target.submit_jobs ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Reimplemented in forcebalance.liquid.Liquid.

Definition at line 288 of file target.py.

#### 8.48.3.11 def forcebalance.target.Target.wq_complete ( *self* )

This method determines whether the Work Queue tasks for the current target have completed.

Definition at line 328 of file target.py.

Here is the call graph for this function:

### 8.48.4 Member Data Documentation

#### 8.48.4.1 forcebalance::target.Target::FF

Need the forcefield (here for now)

Definition at line 106 of file target.py.

#### 8.48.4.2 forcebalance::target.Target::gct

Counts how often the gradient was computed.

Definition at line 108 of file target.py.

#### 8.48.4.3 forcebalance::target.Target::hct

Counts how often the Hessian was computed.

Definition at line 109 of file target.py.

#### 8.48.4.4 forcebalance::target.Target::rundir

The directory in which the simulation is running - this can be updated.

Directory of the current iteration; if not None, then the simulation runs under temp/target_name/iteration_number The 'customdir' is customizable and can go below anything.

Definition at line 105 of file target.py.

#### 8.48.4.5 forcebalance::target.Target::tempdir

Root directory of the whole project.

Name of the target Type of target Relative weight of the target Switch for finite difference gradients Switch for finite difference Hessians Switch for FD gradients + Hessian diagonals How many seconds to sleep (if any) Parameter types that trigger FD gradient elements Parameter types that trigger FD Hessian elements Finite difference step size Whether to make backup files Relative directory of target Temporary (working) directory; it is temp/(target_name) Used for storing temporary variables that don't change through the course of the optimization

Definition at line 104 of file target.py.

#### 8.48.4.6 forcebalance::target.Target::xct

Counts how often the objective function was computed.

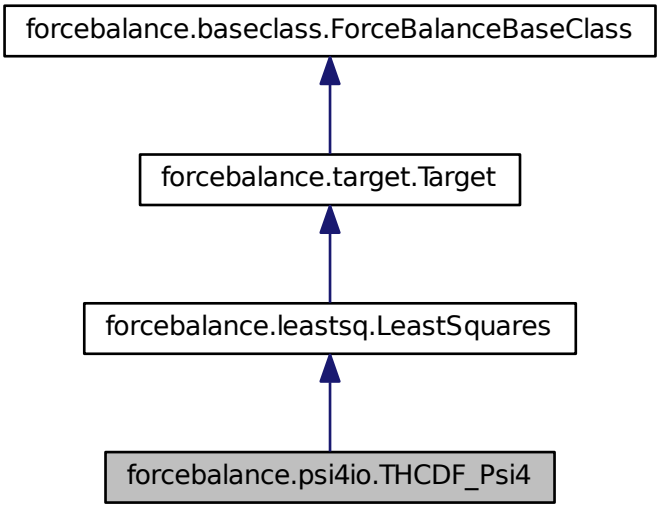
Definition at line 107 of file target.py.

The documentation for this class was generated from the following file:

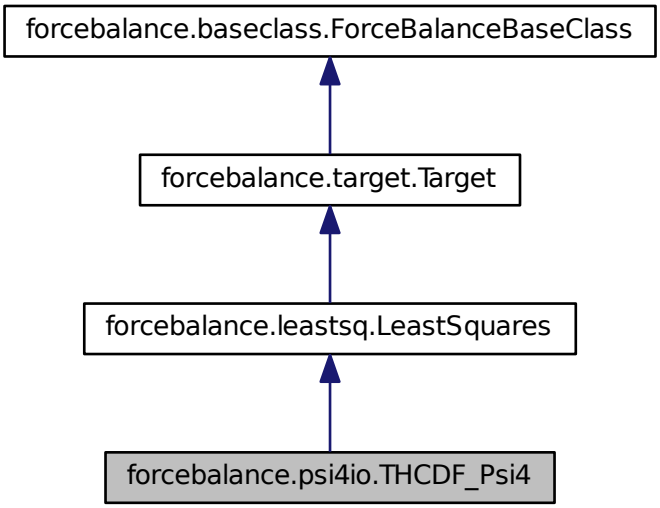
- target.py

## 8.49 forcebalance.psi4io.THCDF_Psi4 Class Reference

Inheritance diagram for forcebalance.psi4io.THCDF_Psi4:




Collaboration diagram for forcebalance.psi4io.THCDF_Psi4:



#### Public Member Functions

- def __init__

  *All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)*
- def prepare_temp_directory
- def indicate
- def write_nested_destroy
- def driver

#### Public Attributes

- Molecules
- throw_outs
- Elements
- GBSfnm

  *Psi4 basis set file.*
- DATfnm

  *Psi4 input file for calculation of linear dependencies This is actually a file in 'forcefield' until we can figure out a better system.*
- MP2_Energy

  *Actually run PSI4.*
- DF_Energy

### 8.49.1 Detailed Description

Definition at line 94 of file psi4io.py.

### 8.49.2 Constructor & Destructor Documentation

#### 8.49.2.1 def forcebalance.psi4io.THCDF_Psi4.__init__ ( *self, options, tgt_opts, forcefield* )

All options here are intended to be usable by every conceivable type of target (in other words, only add content here if it's widely applicable.)

If we want to add attributes that are more specific (i.e. a set of reference forces for force matching), they are added in the subclass AbInitio that inherits from Target.

Reimplemented from forcebalance.leastsq.LeastSquares.

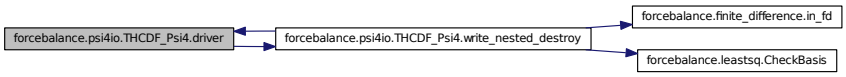
Definition at line 96 of file psi4io.py.

### 8.49.3 Member Function Documentation

#### 8.49.3.1 def forcebalance.psi4io.THCDF_Psi4.driver ( *self* )

Definition at line 171 of file psi4io.py.

Here is the call graph for this function:

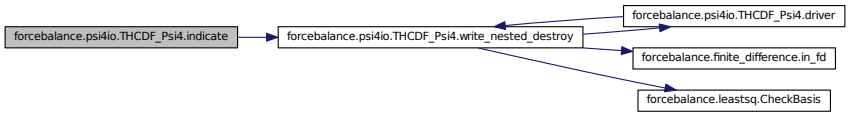



#### 8.49.3.2 def forcebalance.psi4io.THCDF_Psi4.indicate ( *self* )

Reimplemented from forcebalance.leastsq.LeastSquares.

Definition at line 149 of file psi4io.py.

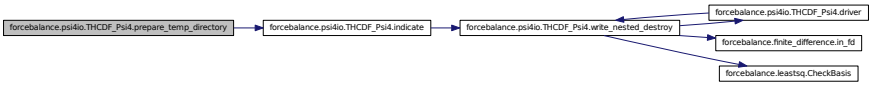
Here is the call graph for this function:




#### 8.49.3.3 def forcebalance.psi4io.THCDF_Psi4.prepare_temp_directory ( *self, options, tgt_opts* )

Definition at line 138 of file psi4io.py.

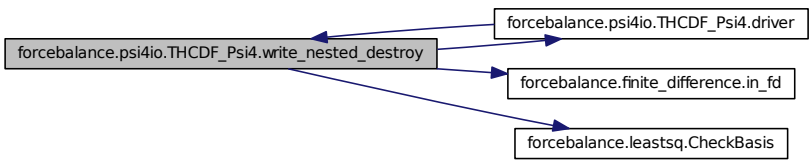
Here is the call graph for this function:




#### 8.49.3.4 def forcebalance.psi4io.THCDF_Psi4.write_nested_destroy ( *self, fnm, linedestroy* )

Definition at line 156 of file psi4io.py.

Here is the call graph for this function:



#### 8.49.4 Member Data Documentation

##### 8.49.4.1 forcebalance::psi4io.THCDF_Psi4::DATfnm

Psi4 input file for calculation of linear dependencies This is actually a file in 'forcefield' until we can figure out a better system.

Definition at line 99 of file psi4io.py.

##### 8.49.4.2 forcebalance::psi4io.THCDF_Psi4::DF_Energy

Definition at line 174 of file psi4io.py.

##### 8.49.4.3 forcebalance::psi4io.THCDF_Psi4::Elements

Definition at line 96 of file psi4io.py.

##### 8.49.4.4 forcebalance::psi4io.THCDF_Psi4::GBSfnm

Psi4 basis set file.

Definition at line 97 of file psi4io.py.

##### 8.49.4.5 forcebalance::psi4io.THCDF_Psi4::Molecules

Definition at line 96 of file psi4io.py.

##### 8.49.4.6 forcebalance::psi4io.THCDF_Psi4::MP2_Energy

Actually run PSI4.

Read in the commented linindep.gbs file and ensure that these same lines are commented in the new .gbs file Now build a "Frankenstein" .gbs file composed of the original .gbs file but with data from the linindep.gbs file!

Definition at line 174 of file psi4io.py.

##### 8.49.4.7 forcebalance::psi4io.THCDF_Psi4::throw_outs

Definition at line 96 of file psi4io.py.

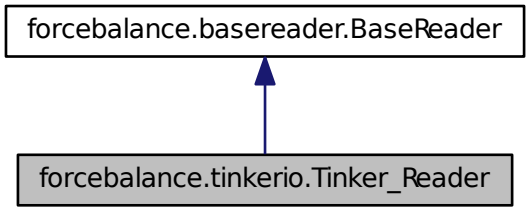
The documentation for this class was generated from the following file:

- psi4io.py

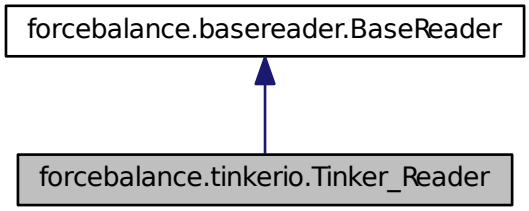
## 8.50 forcebalance.tinkerio.Tinker_Reader Class Reference

Finite state machine for parsing TINKER force field files.

Inheritance diagram for forcebalance.tinkerio.Tinker_Reader:




Collaboration diagram for forcebalance.tinkerio.Tinker_Reader:




#### Public Member Functions

- def __init__
- def feed

  *Given a line, determine the interaction type and the atoms involved (the suffix).*

#### Public Attributes

- pdict

  *The parameter dictionary (defined in this file)*
- atom

  *The atom numbers in the interaction (stored in the TINKER parser)*
- itype
- suffix

### 8.50.1 Detailed Description

Finite state machine for parsing TINKER force field files.

This class is instantiated when we begin to read in a file. The feed(line) method updates the state of the machine, informing it of the current interaction type. Using this information we can look up the interaction type and parameter type for building the parameter ID.

Definition at line 65 of file tinkerio.py.

### 8.50.2 Constructor & Destructor Documentation

#### 8.50.2.1 def forcebalance.tinkerio.Tinker_Reader.__init__ ( *self, fnm* )

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 67 of file tinkerio.py.

### 8.50.3 Member Function Documentation

#### 8.50.3.1 def forcebalance.tinkerio.Tinker_Reader.feed ( *self, line* )

Given a line, determine the interaction type and the atoms involved (the suffix).

TINKER generally has stuff like this:

```
bond-cubic              -2.55
bond-quartic            3.793125

vdw           1               3.4050     0.1100
vdw           2               2.6550     0.0135      0.910 # PARM 4

multipole     2    1    2               0.25983
                                        -0.03859    0.00000   -0.05818
                                        -0.03673
                                         0.00000   -0.10739
                                        -0.00203    0.00000    0.14412
```

The '#PARM 4' has no effect on TINKER but it indicates that we are tuning the fourth field on the line (the 0.910 value).

**Todo** Put the rescaling factors for TINKER parameters in here. Currently we're using the initial value to determine the rescaling factor which is not very good.

Every parameter line is prefaced by the interaction type except for 'multipole' which is on multiple lines. Because the lines that come after 'multipole' are predictable, we just determine the current line using the previous line.

Random note: Unit of force is kcal / mole / angstrom squared.

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 108 of file tinkerio.py.

### 8.50.4 Member Data Documentation

#### 8.50.4.1 forcebalance::tinkerio.Tinker_Reader::atom

The atom numbers in the interaction (stored in the TINKER parser)

Definition at line 69 of file tinkerio.py.

#### 8.50.4.2 forcebalance::tinkerio.Tinker_Reader::itype

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 108 of file tinkerio.py.

#### 8.50.4.3 forcebalance::tinkerio.Tinker_Reader::pdict

The parameter dictionary (defined in this file)

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 68 of file tinkerio.py.

#### 8.50.4.4 forcebalance::tinkerio.Tinker_Reader::suffix

Reimplemented from forcebalance.basereader.BaseReader.

Definition at line 108 of file tinkerio.py.

The documentation for this class was generated from the following file:

- tinkerio.py

## 8.51 forcebalance.nifty.Unpickler_LP Class Reference

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

### Public Member Functions

- def __init__

### 8.51.1 Detailed Description

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

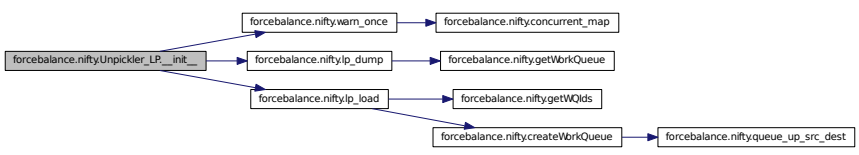
Definition at line 520 of file nifty.py.

### 8.51.2 Constructor & Destructor Documentation

#### 8.51.2.1 def forcebalance.nifty.Unpickler_LP.__init__ ( *self*, *file* )

Definition at line 521 of file nifty.py.

Here is the call graph for this function:



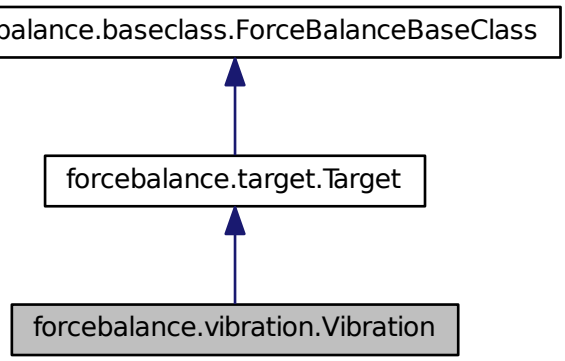
The documentation for this class was generated from the following file:

- nifty.py

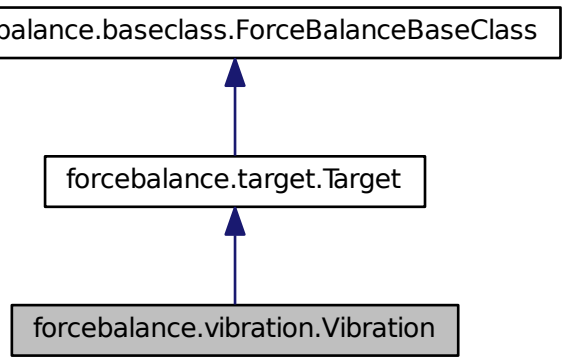
## 8.52 forcebalance.vibration.Vibration Class Reference

Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Inheritance diagram for forcebalance.vibration.Vibration:




Collaboration diagram for forcebalance.vibration.Vibration:



#### Public Member Functions

- def __init__
  *Initialization.*
- def read_reference_data
  *Read the reference vibrational data from a file.*
- def prepare_temp_directory
  *Prepare the temporary directory, by default does nothing.*
- def indicate
  *Print qualitative indicator.*
- def get
  *Evaluate objective function.*

#### Public Attributes

- vfnm
  *The vdata.txt file that contains the vibrations.*
- na
  *Number of atoms.*
- ref_eigvals
- ref_eigvecs
- calc_eigvals
- objective

### 8.52.1 Detailed Description

Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).

Currently Tinker is supported.

Definition at line 27 of file vibration.py.

### 8.52.2 Constructor & Destructor Documentation

#### 8.52.2.1 def forcebalance.vibration.Vibration.__init__ ( *self, options, tgt_opts, forcefield* )

Initialization.

Reimplemented from forcebalance.target.Target.

Definition at line 32 of file vibration.py.

### 8.52.3 Member Function Documentation

#### 8.52.3.1 def forcebalance.vibration.Vibration.get ( *self, mvals, AGrad* = `False`, *AHess* = `False` )

Evaluate objective function.

Reimplemented from forcebalance.target.Target.

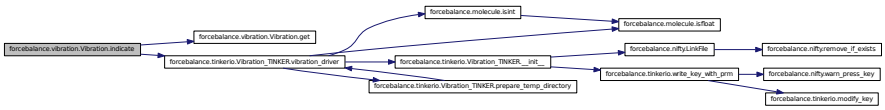
Definition at line 108 of file vibration.py.

#### 8.52.3.2 def forcebalance.vibration.Vibration.indicate ( *self* )

Print qualitative indicator.

Definition at line 99 of file vibration.py.

Here is the call graph for this function:




#### 8.52.3.3 def forcebalance.vibration.Vibration.prepare_temp_directory ( *self, options, tgt_opts* )

Prepare the temporary directory, by default does nothing.

Definition at line 94 of file vibration.py.

#### 8.52.3.4 def forcebalance.vibration.Vibration.read_reference_data ( *self* )

Read the reference vibrational data from a file.

Definition at line 54 of file vibration.py.

### 8.52.4 Member Data Documentation

#### 8.52.4.1 forcebalance::vibration.Vibration::calc_eigvals

Definition at line 108 of file vibration.py.

#### 8.52.4.2 forcebalance::vibration.Vibration::na

Number of atoms.

Definition at line 55 of file vibration.py.

#### 8.52.4.3 forcebalance::vibration.Vibration::objective

Definition at line 108 of file vibration.py.

#### 8.52.4.4 forcebalance::vibration.Vibration::ref_eigvals

Definition at line 55 of file vibration.py.

#### 8.52.4.5 forcebalance::vibration.Vibration::ref_eigvecs

Definition at line 55 of file vibration.py.

#### 8.52.4.6 forcebalance::vibration.Vibration::vfnm

The vdata.txt file that contains the vibrations.

Definition at line 33 of file vibration.py.

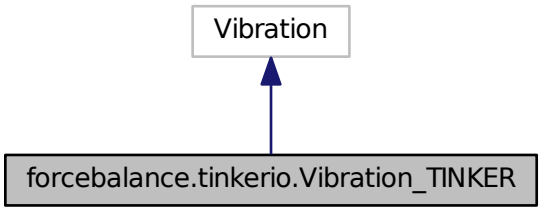
The documentation for this class was generated from the following file:

- vibration.py

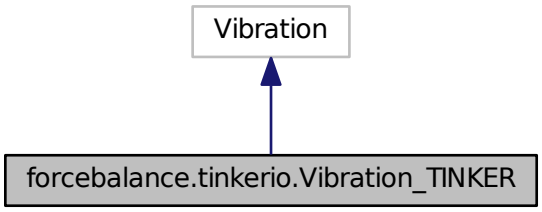
## 8.53 forcebalance.tinkerio.Vibration_TINKER Class Reference

Subclass of Target for vibrational frequency matching using TINKER.

Inheritance diagram for forcebalance.tinkerio.Vibration_TINKER:




Collaboration diagram for forcebalance.tinkerio.Vibration_TINKER:




### Public Member Functions

- def __init__
- def prepare_temp_directory
- def vibration_driver

### 8.53.1 Detailed Description

Subclass of Target for vibrational frequency matching using TINKER.

Provides optimized geometry, vibrational frequencies (in cm-1), and eigenvectors.

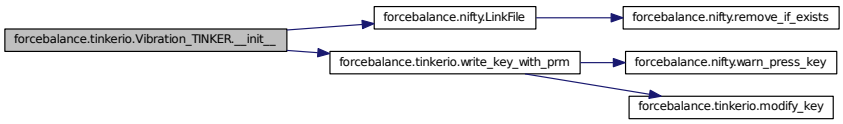
Definition at line 352 of file tinkerio.py.

### 8.53.2 Constructor & Destructor Documentation

#### 8.53.2.1 def forcebalance.tinkerio.Vibration_TINKER.__init__ ( *self, options, tgt_opts, forcefield* )

Definition at line 355 of file tinkerio.py.

Here is the call graph for this function:




### 8.53.3 Member Function Documentation

#### 8.53.3.1 def forcebalance.tinkerio.Vibration_TINKER.prepare_temp_directory ( *self, options, tgt_opts* )

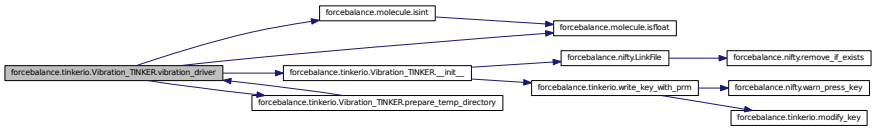
Definition at line 360 of file tinkerio.py.

Here is the call graph for this function:


#### 8.53.3.2 def forcebalance.tinkerio.Vibration_TINKER.vibration_driver ( *self* )

Definition at line 370 of file tinkerio.py.

Here is the call graph for this function:




The documentation for this class was generated from the following file:

- tinkerio.py

# 9 File Documentation

## 9.1 __init__.py File Reference

#### Packages

- namespace forcebalance

## 9.2 abinitio.py File Reference

#### Classes

- class forcebalance.abinitio.AbInitio

  *Subclass of Target for fitting force fields to ab initio data.*

#### Packages

- namespace forcebalance::abinitio

  *Ab-initio fitting module (energies, forces, resp).*

#### Functions

- def forcebalance::abinitio.weighted_variance

  *A more generalized version of build_objective which is callable for derivatives, but the covariance is not there anymore.*
- def forcebalance::abinitio.weighted_variance2

  *A bit of a hack, since we have to subtract out two mean quantities to get Hessian elements.*
- def forcebalance::abinitio.build_objective

  *This function builds an objective function (number) from the complicated polytensor and covariance matrices.*

## 9.3 abinitio_internal.py File Reference

#### Classes

- class forcebalance.abinitio_internal.AbInitio_Internal

  *Subclass of Target for force and energy matching using an internal implementation.*

#### Packages

- namespace forcebalance::abinitio_internal

  *Internal implementation of energy matching (for TIP3P water only)*

## 9.4 amberio.py File Reference

#### Classes

- class forcebalance.amberio.Mol2_Reader

  *Finite state machine for parsing Mol2 force field file.*
- class forcebalance.amberio.FrcMod_Reader

  *Finite state machine for parsing FrcMod force field file.*
- class forcebalance.amberio.AbInitio_AMBER

  *Subclass of Target for force and energy matching using AMBER.*

#### Packages

- namespace forcebalance::amberio

  *AMBER force field input/output.*

#### Functions

- def forcebalance::amberio.is_mol2_atom

#### Variables

- dictionary forcebalance::amberio.mol2_pdict = {'COUL':{'Atom':[1], 8:"}}
- dictionary forcebalance::amberio.frcmod_pdict

## 9.5 api.dox File Reference

## 9.6 baseclass.py File Reference

#### Classes

- class forcebalance.baseclass.ForceBalanceBaseClass

  *Provides some nifty functions that are common to all ForceBalance classes.*

#### Packages

- namespace forcebalance::baseclass

## 9.7 basereader.py File Reference

#### Classes

- class forcebalance.basereader.BaseReader

  *The 'reader' class.*

#### Packages

- namespace forcebalance::basereader

  *Base class for force field line reader.*

## 9.8 binding.py File Reference

#### Classes

- class forcebalance.binding.BindingEnergy

  *Improved subclass of Target for fitting force fields to binding energies.*

#### Packages

- namespace forcebalance::binding

  *Binding energy fitting module.*

#### Functions

- def forcebalance::binding.parse_interactions

  *Parse through the interactions input file.*

## 9.9 chemistry.py File Reference

#### Packages

- namespace forcebalance::chemistry

#### Functions

- def forcebalance::chemistry.LookupByMass
- def forcebalance::chemistry.BondStrengthByLength

#### Variables

- tuple forcebalance::chemistry.BondEnergies = defaultdict(lambda:defaultdict(dict))
- list forcebalance::chemistry.Radii

  *Covalent radii from Cordero et al.*
- dictionary forcebalance::chemistry.PeriodicTable
- list forcebalance::chemistry.Elements
- list forcebalance::chemistry.BondChars = ['-','=','3']
- string forcebalance::chemistry.data_from_web
- tuple forcebalance::chemistry.line = line.expandtabs()
- tuple forcebalance::chemistry.BE = float(line.split()[1])
- tuple forcebalance::chemistry.L = float(line.split()[2])
- tuple forcebalance::chemistry.atoms = re.split('[-=3]', line.split()[0])
- list forcebalance::chemistry.A = atoms[0]
- list forcebalance::chemistry.B = atoms[1]
- tuple forcebalance::chemistry.bo = BondChars.index(re.findall('[-=3]', line.split()[0])[0])

## 9.10 contact.py File Reference

#### Packages

- namespace forcebalance::contact

#### Functions

- def forcebalance::contact.atom_distances

  *For each frame in xyzlist, compute the (euclidean) distance between pairs of atoms whos indices are given in contacts.*
- def forcebalance::contact.residue_distances

  *For each frame in xyzlist, and for each pair of residues in the array contact, compute the distance between the closest pair of atoms such that one of them belongs to each residue.*

## 9.11 counterpoise.py File Reference

#### Classes

- class forcebalance.counterpoise.Counterpoise

  *Target subclass for matching the counterpoise correction.*

#### Packages

- namespace forcebalance::counterpoise

  *Match an empirical potential to the counterpoise correction for basis set superposition error (BSSE).*

## 9.12 custom_io.py File Reference

#### Classes

- class forcebalance.custom_io.Gen_Reader

  *Finite state machine for parsing custom GROMACS force field files.*

#### Packages

- namespace forcebalance::custom_io

  *Custom force field parser.*

#### Variables

- list forcebalance::custom_io.cptypes = [None, 'CPGAUSS', 'CPEXPG', 'CPGEXP']

  *Types of counterpoise correction.*
- list forcebalance::custom_io.ndtypes = [None]

  *Types of NDDO correction.*
- dictionary forcebalance::custom_io.fdict

  *Section -> Interaction type dictionary.*
- dictionary forcebalance::custom_io.pdict

  *Interaction type -> Parameter Dictionary.*

## 9.13 finite_difference.py File Reference

#### Packages

- namespace forcebalance::finite_difference

### Functions

- def forcebalance::finite_difference.f1d2p

  *A two-point finite difference stencil.*
- def forcebalance::finite_difference.f1d5p

  *A highly accurate five-point finite difference stencil for computing derivatives of a function.*
- def forcebalance::finite_difference.f1d7p

  *A highly accurate seven-point finite difference stencil for computing derivatives of a function.*
- def forcebalance::finite_difference.f12d7p
- def forcebalance::finite_difference.f12d3p

  *A three-point finite difference stencil.*
- def forcebalance::finite_difference.in_fd

  *Invoking this function from anywhere will tell us whether we're being called by a finite-difference function.*
- def forcebalance::finite_difference.fdwrap

  *A function wrapper for finite difference designed for differentiating 'get'-type functions.*
- def forcebalance::finite_difference.fdwrap_G

  *A driver to fdwrap for gradients (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.*
- def forcebalance::finite_difference.fdwrap_H

  *A driver to fdwrap for Hessians (see documentation for fdwrap) Inputs: tgt = The Target containing the objective function that we want to differentiate mvals0 = The 'central' values of the mathematical parameters - i.e.*

## 9.14 forcefield.py File Reference

### Classes

- class forcebalance.forcefield.BackedUpDict
- class forcebalance.forcefield.FF

  *Force field class.*

### Packages

- namespace forcebalance::forcefield

  *Force field module.*

### Functions

- def forcebalance::forcefield.determine_fftype

  *Determine the type of a force field file.*
- def forcebalance::forcefield.rs_override

  *This function takes in a dictionary (rsfactors) and a string (termtype).*

### Variables

- dictionary forcebalance::forcefield.FF_Extensions
- dictionary forcebalance::forcefield.FF_IOModules

## 9.15 gmxio.py File Reference

### Classes

- class forcebalance.gmxio.ITP_Reader

  *Finite state machine for parsing GROMACS force field files.*
- class forcebalance.gmxio.AbInitio_GMX

  *Subclass of AbInitio for force and energy matching using normal GROMACS.*
- class forcebalance.gmxio.Liquid_GMX
- class forcebalance.gmxio.Interaction_GMX

  *Subclass of Interaction for interaction energy matching using GROMACS.*

### Packages

- namespace forcebalance::gmxio

  *GROMACS input/output.*

### Functions

- def forcebalance::gmxio.edit_mdp

  *Create or edit a Gromacs MDP file.*
- def forcebalance::gmxio.parse_atomtype_line

  *Parses the 'atomtype' line.*
- def forcebalance::gmxio.rm_gmx_baks

### Variables

- list forcebalance::gmxio.nftypes = [None, 'VDW', 'VDW_BHAM']

  *VdW interaction function types.*
- list forcebalance::gmxio.pftypes = [None, 'VPAIR', 'VPAIR_BHAM']

  *Pairwise interaction function types.*
- list forcebalance::gmxio.bftypes = [None, 'BONDS', 'G96BONDS', 'MORSE']

  *Bonded interaction function types.*
- list forcebalance::gmxio.aftypes

  *Angle interaction function types.*
- list forcebalance::gmxio.dftypes = [None, 'PDIHS', 'IDIHS', 'RBDIHS', 'PIMPDIHS', 'FOURDIHS', None, None, 'TABDIHS', 'PDIHMULS']

  *Dihedral interaction function types.*
- dictionary forcebalance::gmxio.fdict

  *Section -> Interaction type dictionary.*
- dictionary forcebalance::gmxio.pdict

  *Interaction type -> Parameter Dictionary.*

## 9.16 gmxqpio.py File Reference

### Classes

- class forcebalance.gmxqpio.Monomer_QTPIE

  *Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).*

#### Packages

- namespace forcebalance::gmxqpio
- namespace forcebalance::gmxio

  *GROMACS input/output.*

#### Functions

- def forcebalance::gmxqpio.get_monomer_properties

## 9.17 interaction.py File Reference

#### Classes

- class forcebalance.interaction.Interaction

  *Subclass of Target for fitting force fields to interaction energies.*

#### Packages

- namespace forcebalance::interaction

  *Interaction energy fitting module.*

## 9.18 leastsq.py File Reference

#### Classes

- class forcebalance.leastsq.LeastSquares

  *Subclass of Target for general least squares fitting.*

#### Packages

- namespace forcebalance::leastsq
- namespace forcebalance::abinitio

  *Ab-initio fitting module (energies, forces, resp).*

#### Functions

- def forcebalance::leastsq.CheckBasis
- def forcebalance::leastsq.LastMvals

#### Variables

- forcebalance::leastsq.CHECK_BASIS = False
- forcebalance::leastsq.LAST_MVALS = None

## 9.19 liquid.py File Reference

### Classes

- class forcebalance.liquid.Liquid

  *Subclass of Target for liquid property matching.*

### Packages

- namespace forcebalance::liquid

  *Matching of liquid bulk properties.*

### Functions

- def forcebalance::liquid.weight_info

## 9.20 Mol2.py File Reference

### Classes

- class forcebalance.Mol2.mol2_atom

  *This is to manage mol2 atomic lines on the form: 1 C1 5.4790 42.2880 49.5910 C.ar 1 <1> 0.0424.*
- class forcebalance.Mol2.mol2_bond

  *This is to manage mol2 bond lines on the form: 1 1 2 ar.*
- class forcebalance.Mol2.mol2

  *This is to manage one mol2 series of lines on the form:*
- class forcebalance.Mol2.mol2_set

### Packages

- namespace forcebalance::Mol2

### Variables

- tuple forcebalance::Mol2.data = mol2_set(sys.argv[1], subset=["RNAse.xray.inh8.1QHC"])

## 9.21 mol2io.py File Reference

### Classes

- class forcebalance.mol2io.Mol2_Reader

  *Finite state machine for parsing Mol2 force field file.*

### Packages

- namespace forcebalance::mol2io

  *Mol2 I/O.*

#### Variables

- dictionary forcebalance::mol2io.mol2_pdict = {'COUL':{'Atom':[1], 6:''}}

## 9.22 molecule.py File Reference

#### Classes

- class forcebalance.molecule.MolfileTimestep

  *Wrapper for the timestep C structure used in molfile plugins.*
- class forcebalance.molecule.Molecule

  *Lee-Ping's general file format conversion class.*

#### Packages

- namespace forcebalance::molecule

#### Functions

- def forcebalance::molecule.getElement
- def forcebalance::molecule.nodematch
- def forcebalance::molecule.isint

  *ONLY matches integers! If you have a decimal point? None shall pass!*
- def forcebalance::molecule.isfloat

  *Matches ANY number; it can be a decimal, scientific notation, integer, or what have you.*
- def forcebalance::molecule.BuildLatticeFromLengthsAngles

  *This function takes in three lattice lengths and three lattice angles, and tries to return a complete box specification.*
- def forcebalance::molecule.BuildLatticeFromVectors

  *This function takes in three lattice vectors and tries to return a complete box specification.*
- def forcebalance::molecule.format_xyz_coord

  *Print a line consisting of (element, x, y, z) in accordance with .xyz file format.*
- def forcebalance::molecule.format_gro_coord

  *Print a line in accordance with .gro file format, with six decimal points of precision.*
- def forcebalance::molecule.format_xyzgen_coord

  *Print a line consisting of (element, p, q, r, s, t, ...) where (p, q, r) are arbitrary atom-wise data (this might happen, for instance, with atomic charges)*
- def forcebalance::molecule.format_gro_box

  *Print a line corresponding to the box vector in accordance with .gro file format.*
- def forcebalance::molecule.is_gro_coord

  *Determines whether a line contains GROMACS data or not.*
- def forcebalance::molecule.is_charmm_coord

  *Determines whether a line contains CHARMM data or not.*
- def forcebalance::molecule.is_gro_box

  *Determines whether a line contains a GROMACS box vector or not.*
- def forcebalance::molecule.add_strip_to_mat
- def forcebalance::molecule.pvec
- def forcebalance::molecule.grouper

*Groups a big long iterable into groups of ten or what have you.*

- def forcebalance::molecule.even_list

  *Creates a list of number sequences divided as evenly as possible.*

- def forcebalance::molecule.both
- def forcebalance::molecule.diff
- def forcebalance::molecule.either
- def forcebalance::molecule.EulerMatrix

  *Constructs an Euler matrix from three Euler angles.*

- def forcebalance::molecule.ComputeOverlap

  *Computes an 'overlap' between two molecules based on some fictitious density.*

- def forcebalance::molecule.AlignToDensity

  *Computes a "overlap density" from two frames.*

- def forcebalance::molecule.AlignToMoments

  *Pre-aligns molecules to 'moment of inertia'.*

- def forcebalance::molecule.get_rotate_translate
- def forcebalance::molecule.main

#### Variables

- tuple forcebalance::molecule::FrameVariableNames
- tuple forcebalance::molecule.AtomVariableNames = set(['elem', 'partial_charge', 'atomname', 'atomtype', 'tinkersuf', 'resid', 'resname', 'qcsuf', 'qm_ghost', 'chain', 'altloc', 'icode'])
- tuple forcebalance::molecule.MetaVariableNames = set(['fnm', 'ftype', 'qcrems', 'qctemplate', 'charge', 'mult', 'bonds'])
- tuple forcebalance::molecule.QuantumVariableNames = set(['qcrems', 'qctemplate', 'charge', 'mult', 'qcsuf', 'qm_ghost'])
- forcebalance::molecule.AllVariableNames = QuantumVariableNames|AtomVariableNames|MetaVariableNames|FrameVariableNames
- list forcebalance::molecule.Radii
- list forcebalance::molecule.Elements
- tuple forcebalance::molecule.PeriodicTable
- float forcebalance::molecule.bohrang = 0.529177249

  *One bohr equals this many angstroms.*

- tuple forcebalance::molecule.splitter = re.compile(r'(\s+|\S+)')
- tuple forcebalance::molecule.Box = namedtuple('Box',['a','b','c','alpha','beta','gamma','A','B','C','V'])
- int forcebalance::molecule.radian = 180
- forcebalance::molecule::Alive

## 9.23 moments.py File Reference

#### Classes

- class forcebalance.moments.Moments

  *Subclass of Target for fitting force fields to multipole moments (from experiment or theory).*

#### Packages

- namespace forcebalance::moments

  *Multipole moment fitting module.*

## 9.24 nifty.py File Reference

### Classes

- class forcebalance.nifty.Pickler_LP

  *A subclass of the python Pickler that implements pickling of _ElementTree types.*
- class forcebalance.nifty.Unpickler_LP

  *A subclass of the python Unpickler that implements unpickling of _ElementTree types.*
- class forcebalance.nifty.RawStreamHandler

  *Exactly like logging.StreamHandler except it does no extra formatting before sending logging messages to the stream.*
- class forcebalance.nifty.RawFileHandler

  *Exactly like logging.FileHandler except it does no extra formatting before sending logging messages to the file.*

### Packages

- namespace forcebalance::nifty

  *Nifty functions, intended to be imported by any module within ForceBalance.*

### Functions

- def forcebalance::nifty.pvec1d

  *Printout of a 1-D vector.*
- def forcebalance::nifty.pmat2d

  *Printout of a 2-D matrix.*
- def forcebalance::nifty.encode
- def forcebalance::nifty.segments
- def forcebalance::nifty.commadash
- def forcebalance::nifty.uncommadash
- def forcebalance::nifty.printcool

  *Cool-looking printout for slick formatting of output.*
- def forcebalance::nifty.printcool_dictionary

  *See documentation for printcool; this is a nice way to print out keys/values in a dictionary.*
- def forcebalance::nifty.isint

  *ONLY matches integers! If you have a decimal point? None shall pass!*
- def forcebalance::nifty.isfloat

  *Matches ANY number; it can be a decimal, scientific notation, what have you CAUTION - this will also match an integer.*
- def forcebalance::nifty.isdecimal

  *Matches things with a decimal only; see isint and isfloat.*
- def forcebalance::nifty.floatornan

  *Returns a big number if we encounter NaN.*
- def forcebalance::nifty.col

  *Given any list, array, or matrix, return a 1-column matrix.*
- def forcebalance::nifty.row

  *Given any list, array, or matrix, return a 1-row matrix.*
- def forcebalance::nifty.flat

  *Given any list, array, or matrix, return a single-index array.*
- def forcebalance::nifty.orthogonalize

*Given two vectors vec1 and vec2, project out the component of vec1 that is along the vec2-direction.*

- def forcebalance::nifty.invert_svd

  *Invert a matrix using singular value decomposition.*
- def forcebalance::nifty.get_least_squares
- def forcebalance::nifty.statisticalInefficiency

  *Compute the (cross) statistical inefficiency of (two) timeseries.*
- def forcebalance::nifty.lp_dump

  *Use this instead of pickle.dump for pickling anything that contains _ElementTree types.*
- def forcebalance::nifty.lp_load

  *Use this instead of pickle.load for unpickling anything that contains _ElementTree types.*
- def forcebalance::nifty.getWorkQueue
- def forcebalance::nifty.getWQIds
- def forcebalance::nifty.createWorkQueue
- def forcebalance::nifty.queue_up

  *Submit a job to the Work Queue.*
- def forcebalance::nifty.queue_up_src_dest

  *Submit a job to the Work Queue.*
- def forcebalance::nifty.wq_wait1

  *This function waits ten seconds to see if a task in the Work Queue has finished.*
- def forcebalance::nifty.wq_wait

  *This function waits until the work queue is completely empty.*
- def forcebalance::nifty.GoInto
- def forcebalance::nifty.allsplit
- def forcebalance::nifty.Leave
- def forcebalance::nifty.MissingFileInspection
- def forcebalance::nifty.LinkFile
- def forcebalance::nifty.CopyFile
- def forcebalance::nifty.link_dir_contents
- def forcebalance::nifty.remove_if_exists

  *Remove the file if it exists (doesn't return an error).*
- def forcebalance::nifty.which
- def forcebalance::nifty.warn_press_key
- def forcebalance::nifty.warn_once

  *Prints a warning but will only do so once in a given run.*
- def forcebalance::nifty.concurrent_map

  *Similar to the bultin function map().*
- def forcebalance::nifty.multiopen

  *This function be given any of several variable types (single file name, file object, or list of lines, or a list of the above) and give a list of files:*

### Variables

- float forcebalance::nifty.kb = 0.0083144100163

  *Boltzmann constant.*
- float forcebalance::nifty.eqcgmx = 2625.5002

  *Q-Chem to GMX unit conversion for energy.*
- float forcebalance::nifty.fqcgmx = 49621.9

  *Q-Chem to GMX unit conversion for force.*

- float forcebalance::nifty.bohrang = 0.529177249

  *One bohr equals this many angstroms.*
- string forcebalance::nifty.XMLFILE = 'x'

  *Pickle uses 'flags' to pickle and unpickle different variable types.*
- forcebalance::nifty.WORK_QUEUE = None
- tuple forcebalance::nifty.WQIDS = defaultdict(list)
- list forcebalance::nifty.specific_lst
- tuple forcebalance::nifty.specific_dct = dict(list(itertools.chain(*[[(j,i[1]) for j in i[0]] for i in specific_lst])))

## 9.25 objective.py File Reference

### Classes

- class forcebalance.objective.Objective

  *Objective function.*
- class forcebalance.objective.Penalty

  *Penalty functions for regularizing the force field optimizer.*

### Packages

- namespace forcebalance::objective

  *ForceBalance objective function.*

### Variables

- dictionary forcebalance::objective.Implemented_Targets

  *The table of implemented Targets.*
- list forcebalance::objective.Letters = ['X','G','H']

  *This is the canonical lettering that corresponds to : objective function, gradient, Hessian.*

## 9.26 openmmio.py File Reference

### Classes

- class forcebalance.openmmio.OpenMM_Reader

  *Class for parsing OpenMM force field files.*
- class forcebalance.openmmio.Liquid_OpenMM
- class forcebalance.openmmio.AbInitio_OpenMM

  *Subclass of AbInitio for force and energy matching using OpenMM.*
- class forcebalance.openmmio.Interaction_OpenMM

  *Subclass of Target for interaction matching using OpenMM.*

### Packages

- namespace forcebalance::openmmio

  *OpenMM input/output.*

#### Functions

- def forcebalance::openmmio.get_dipole

  *Return the current dipole moment in Debye.*
- def forcebalance::openmmio.ResetVirtualSites

  *Given a set of OpenMM-compatible positions and a System object, compute the correct virtual site positions according to the System.*
- def forcebalance::openmmio.CopyAmoebaBondParameters
- def forcebalance::openmmio.CopyAmoebaOutOfPlaneBendParameters
- def forcebalance::openmmio.CopyAmoebaAngleParameters
- def forcebalance::openmmio.CopyAmoebaInPlaneAngleParameters
- def forcebalance::openmmio.CopyAmoebaVdwParameters
- def forcebalance::openmmio.CopyAmoebaMultipoleParameters
- def forcebalance::openmmio.CopyHarmonicBondParameters
- def forcebalance::openmmio.CopyHarmonicAngleParameters
- def forcebalance::openmmio.CopyPeriodicTorsionParameters
- def forcebalance::openmmio.CopyNonbondedParameters
- def forcebalance::openmmio.do_nothing
- def forcebalance::openmmio.CopySystemParameters

  *Copy parameters from one system (i.e.*
- def forcebalance::openmmio.UpdateSimulationParameters
- def forcebalance::openmmio.MTSVVVRIntegrator

  *Create a multiple timestep velocity verlet with velocity randomization (VVVR) integrator.*

#### Variables

- dictionary forcebalance::openmmio.suffix_dict
- string forcebalance::openmmio.pdict = "XML_Override"

  *pdict is a useless variable if the force field is XML.*

## 9.27 optimizer.py File Reference

#### Classes

- class forcebalance.optimizer.Optimizer

  *Optimizer class.*

#### Packages

- namespace forcebalance::optimizer

  *Optimization algorithms.*

#### Functions

- def forcebalance::optimizer.Counter
- def forcebalance::optimizer.GoodStep

#### Variables

- int forcebalance::optimizer.ITERATION_NUMBER = 0
- int forcebalance::optimizer.GOODSTEP = 0

## 9.28 parser.py File Reference

#### Packages

- namespace forcebalance::parser

  *Input file parser for ForceBalance jobs.*

#### Functions

- def forcebalance::parser.read_mvals
- def forcebalance::parser.read_pvals
- def forcebalance::parser.read_priors
- def forcebalance::parser.read_internals
- def forcebalance::parser.printsection

  *Print out a section of the input file in a parser-compliant and readable format.*
- def forcebalance::parser.parse_inputs

  *Parse through the input file and read all user-supplied options.*

#### Variables

- dictionary forcebalance::parser.gen_opts_types

  *Default general options.*
- dictionary forcebalance::parser.tgt_opts_types

  *Default fitting target options.*
- dictionary forcebalance::parser.gen_opts_defaults = {}

  *Default general options - basically a collapsed veresion of gen_opts_types.*
- dictionary forcebalance::parser.subdict = {}
- dictionary forcebalance::parser.tgt_opts_defaults = {}

  *Default target options - basically a collapsed version of tgt_opts_types.*
- dictionary forcebalance::parser.bkwd = {"simtype" : "type"}

  *Option maps for maintaining backward compatibility.*
- list forcebalance::parser.mainsections = ["SIMULATION","TARGET","OPTIONS","END","NONE"]

  *Listing of sections in the input file.*
- dictionary forcebalance::parser.ParsTab

  *ParsTab that refers to subsection parsers.*

## 9.29 psi4io.py File Reference

#### Classes

- class forcebalance.psi4io.GBS_Reader

  *Interaction type -> Parameter Dictionary.*

- class forcebalance.psi4io.THCDF_Psi4
- class forcebalance.psi4io.Grid_Reader

  *Finite state machine for parsing DVR grid files.*
- class forcebalance.psi4io.RDVR3_Psi4

  *Subclass of Target for R-DVR3 grid fitting.*

#### Packages

- namespace forcebalance::psi4io

  *PSI4 force field input/output.*

## 9.30 PT.py File Reference

#### Packages

- namespace forcebalance::PT

#### Variables

- dictionary forcebalance::PT::PeriodicTable
- list forcebalance::PT.Elements

## 9.31 qchemio.py File Reference

#### Classes

- class forcebalance.qchemio.QCIn_Reader

  *Finite state machine for parsing Q-Chem input files.*

#### Packages

- namespace forcebalance::qchemio

  *Q-Chem input file parser.*

#### Functions

- def forcebalance::qchemio.QChem_Dielectric_Energy

#### Variables

- list forcebalance::qchemio.ndtypes = [None]

  *Types of counterpoise correction cptypes = [None, 'BASS', 'BASSP'] Types of NDDO correction.*
- dictionary forcebalance::qchemio.pdict

  *Section -> Interaction type dictionary.*

## 9.32 target.py File Reference

### Classes

- class forcebalance.target.Target

  *Base class for all fitting targets.*

### Packages

- namespace forcebalance::target

## 9.33 tinkerio.py File Reference

### Classes

- class forcebalance.tinkerio.Tinker_Reader

  *Finite state machine for parsing TINKER force field files.*
- class forcebalance.tinkerio.Liquid_TINKER
- class forcebalance.tinkerio.AbInitio_TINKER

  *Subclass of Target for force and energy matching using TINKER.*
- class forcebalance.tinkerio.Vibration_TINKER

  *Subclass of Target for vibrational frequency matching using TINKER.*
- class forcebalance.tinkerio.Moments_TINKER

  *Subclass of Target for multipole moment matching using TINKER.*
- class forcebalance.tinkerio.BindingEnergy_TINKER

  *Subclass of BindingEnergy for binding energy matching using TINKER.*
- class forcebalance.tinkerio.Interaction_TINKER

  *Subclass of Target for interaction matching using TINKER.*

### Packages

- namespace forcebalance::tinkerio

  *TINKER input/output.*

### Functions

- def forcebalance::tinkerio.write_key_with_prm

  *Copies a TINKER .key file but changes the parameter keyword as necessary to reflect the ForceBalance settings.*
- def forcebalance::tinkerio.modify_key

  *Performs in-place modification of a TINKER .key file.*

### Variables

- dictionary forcebalance::tinkerio.pdict

## 9.34 vibration.py File Reference

### Classes

- class forcebalance.vibration.Vibration

  *Subclass of Target for fitting force fields to vibrational spectra (from experiment or theory).*

### Packages

- namespace forcebalance::vibration

  *Vibrational mode fitting module.*