ForceBalance Developer API Guide version 1.1

Generated by Doxygen 1.7.6.1

Contents

1 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_covariance_

Parallelization over snapshots is not implemented yet

Member forcebalance.abinitio.AbInitio.get_energy_force_no_covariance_

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

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.

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.

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.

2 Namespace Index

2.1 Packages

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

3 Hierarchical Index

3.1 Class Hierarchy

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

4 Class Index

4.1 Class List

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

5 Namespace Documentation

5.1 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.

5.1.1 Detailed Description

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

Author

Lee-Ping Wang

Date

05/2012

5.1.2 Function Documentation

5.1.2.1 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.

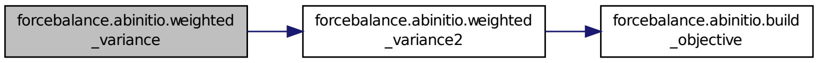
Definition at line 1257 of file abinitio.py.

5.1.2.2 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 1227 of file abinitio.py.

Here is the call graph for this function:

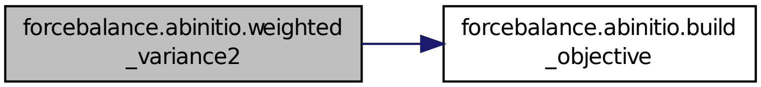

forcebalance.abinitio.weighted _variance
forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.build _objective

5.1.2.3 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 1241 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.build _objective

5.2 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.

5.2.1 Detailed Description

Internal implementation of energy matching (for TIP3P water only)

Author

Lee-Ping Wang

Date

04/2012

5.3 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**

5.3.1 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

5.3.2 Variable Documentation

5.3.2.1 dictionary forcebalance.amberio.frcmod_pdict

Initial value:

```
= {'BONDS': {'Atom':[0], 1:'K', 2:'B'},
                 'ANGLES':{'Atom':[0], 1:'K', 2:'B'},
                 'PDIHS1':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS2':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS3':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS4':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS5':{'Atom':[0], 2:'K', 3:'B'},
                 'PDIHS6':{'Atom':[0], 2:'K', 3:'B'},
                 'IDIHS' :{'Atom':[0], 1:'K', 3:'B'},
                  'VDW':{'Atom':[0], 1:'S', 2:'T'}
                  }
```

Definition at line 20 of file amberio.py.

5.4 forcebalance.basereader Namespace Reference

Base class for force field line reader.

Classes

- class BaseReader

  The 'reader' class.

5.4.1 Detailed Description

Base class for force field line reader.

Author

Lee-Ping Wang

Date

12/2011

5.5 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.

5.5.1 Detailed Description

Binding energy fitting module.

Author

Lee-Ping Wang

Date

05/2012

5.5.2 Function Documentation

5.5.2.1 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.

5.6 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.

5.6.1 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

5.7 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.

5.7.1 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

5.7.2 Variable Documentation

5.7.2.1 dictionary forcebalance.custom_io.fdict

Initial value:

```
= {
   'counterpoise'  : cptypes    }
```

Section -> Interaction type dictionary.

Definition at line 21 of file custom_io.py.

5.7.2.2 dictionary forcebalance.custom_io.pdict

Initial value:

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

Interaction type -> Parameter Dictionary.

Definition at line 25 of file custom_io.py.

5.8 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**

5.8.1 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

5.8.2 Function Documentation

5.8.2.1 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 144 of file forcefield.py.

5.8.2.2 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 1153 of file forcefield.py.

5.8.3 Variable Documentation

5.8.3.1 dictionary forcebalance.forcefield.FF_Extensions

Initial value:

```
= {"itp" : "gmx",
                   "in"  : "qchem",
                   "prm" : "tinker",
                   "gen" : "custom",
                   "xml" : "openmm",
                   "frcmod" : "frcmod",
                   "mol2" : "mol2",
                   "gbs"  : "gbs",
                   "grid" : "grid"
                    }
```

Definition at line 116 of file forcefield.py.

5.8.3.2 dictionary forcebalance.forcefield.FF_IOModules

Initial value:

```
= {"gmx": gmxio.ITP_Reader ,
                  "qchem": qchemio.QCIn_Reader ,
                  "tinker": tinkerio.Tinker_Reader ,
                  "custom": custom_io.Gen_Reader ,
                  "openmm" : openmmio.OpenMM_Reader,
                  "frcmod" : amberio.FrcMod_Reader,
                  "mol2" : amberio.Mol2_Reader,
                  "gbs" : psi4io.GBS_Reader,
                  "grid" : psi4io.Grid_Reader
                   }
```

Definition at line 128 of file forcefield.py.

5.9 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 Interaction_GMX

  Subclass of Interaction for interaction energy matching using GROMACS.

Functions

- 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', 'PDIHMU-LS']

  Dihedral interaction function types.
- dictionary fdict

  Section -> Interaction type dictionary.
- dictionary pdict

  Interaction type -> Parameter Dictionary.

5.9.1 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

5.9.2 Function Documentation

5.9.2.1 def forcebalance.gmxio.parse_atomtype_line (*line*)

Parses the 'atomtype' line.

```
Parses lines like this:\n
<tt> opls_135     CT    6   12.0107    0.0000    A    3.5000e-01    2.7614e-01\n
C       12.0107    0.0000    A    3.7500e-01    4.3932e-01\n
Na   11    22.9897    0.0000    A    6.068128070229e+03  2.662662556402e+01  0.0000e+00 ; PARM 5 6\n </tt>
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 120 of file gmxio.py.

5.9.3 Variable Documentation

5.9.3.1 list forcebalance.gmxio.aftypes

Initial value:

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

Angle interaction function types.

Definition at line 30 of file gmxio.py.

5.9.3.2 dictionary forcebalance.gmxio.fdict

Initial value:

```
= {
    'atomtypes'     : nftypes,
    'nonbond_params': pftypes,
    'bonds'         : bftypes,
    'bondtypes'     : bftypes,
    'angles'        : aftypes,
    'angletypes'    : aftypes,
    'dihedrals'     : dftypes,
    'dihedraltypes' : dftypes,
     'virtual_sites2': ['NONE','VSITE2'],
     'virtual_sites3': ['NONE','VSITE3','VSITE3FD','VSITE3FAD','VSITE3OUT'],
     'virtual_sites4': ['NONE','VSITE4FD','VSITE4FDN']
     }
```

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 41 of file gmxio.py.

5.9.3.3 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 64 of file gmxio.py.

5.10 forcebalance.implemented Namespace Reference

Contains the dictionary of usable Target classes.

Variables

- dictionary Implemented_Targets

  The table of implemented Targets.

5.10.1 Detailed Description

Contains the dictionary of usable Target classes.

5.10.2 Variable Documentation

5.10.2.1 dictionary forcebalance.implemented.Implemented_Targets

Initial value:

```
= {
    'ABINITIO_GMX':AbInitio_GMX,
    'ABINITIO_TINKER':AbInitio_TINKER,
    'ABINITIO_OPENMM':AbInitio_OpenMM,
    'ABINITIO_AMBER':AbInitio_AMBER,
    'ABINITIO_INTERNAL':AbInitio_Internal,
    'VIBRATION_TINKER':Vibration_TINKER,
    'LIQUID_OPENMM':Liquid_OpenMM,
    'LIQUID_TINKER':Liquid_TINKER,
     'COUNTERPOISE':Counterpoise,
     'THCDF_PSI4':THCDF_Psi4,
     'RDVR3_PSI4':RDVR3_Psi4,
     'INTERACTION_TINKER':Interaction_TINKER,
     'INTERACTION_OPENMM':Interaction_OpenMM,
     'BINDINGENERGY_TINKER':BindingEnergy_TINKER,
     'MOMENTS_TINKER':Moments_TINKER,
     'MONOMER_QTPIE':Monomer_QTPIE,
     }
```

The table of implemented Targets.

Definition at line 55 of file implemented.py.

5.11 forcebalance.interaction Namespace Reference

Interaction energy fitting module.

Classes

- class Interaction

    Subclass of Target for fitting force fields to interaction energies.

5.11.1 Detailed Description

Interaction energy fitting module.

Author

Lee-Ping Wang

Date

05/2012

5.12 forcebalance.liquid Namespace Reference

Matching of liquid bulk properties.

Classes

- class Liquid

    Subclass of Target for liquid property matching.

Functions

- def **weight_info**

5.12.1 Detailed Description

Matching of liquid bulk properties. Under development.

Author

Lee-Ping Wang

Date

04/2012

5.13 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:''}}

5.13.1 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

5.14 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).

5.14.1 Detailed Description

Multipole moment fitting module.

Author

Lee-Ping Wang

Date

09/2012

5.15 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.

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 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])))

5.15.1 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

5.15.2 Function Documentation

5.15.2.1 def forcebalance.nifty.col (*vec*)

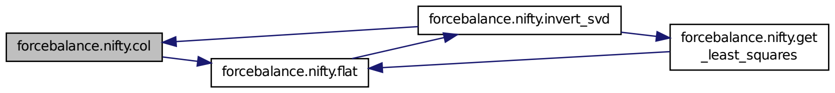
Given any list, array, or matrix, return a 1-column matrix.

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

Output: A column matrix

Definition at line 254 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.col
forcebalance.nifty.invert_svd
forcebalance.nifty.flat
forcebalance.nifty.get _least_squares

5.15.2.2 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 785 of file nifty.py.

5.15.2.3 def forcebalance.nifty.flat (*vec*)

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

Parameters

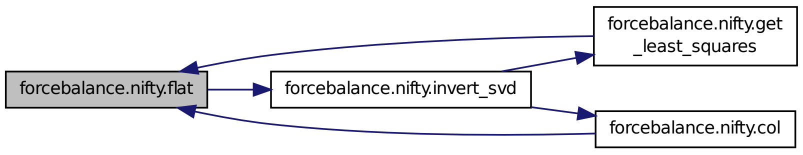
in	*vec*	The data to be flattened

Returns

answer The flattened data

Definition at line 273 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.flat
forcebalance.nifty.get_least_squares
forcebalance.nifty.invert_svd
forcebalance.nifty.col

5.15.2.4 def forcebalance.nifty.floatornan (*word*)

Returns a big number if we encounter NaN.

Parameters

in	*word*	The string to be converted

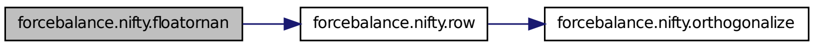
Returns

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 236 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

5.15.2.5 def forcebalance.nifty.get_least_squares (*x*, *y*, *w* = None, *thresh* = 1e−12)

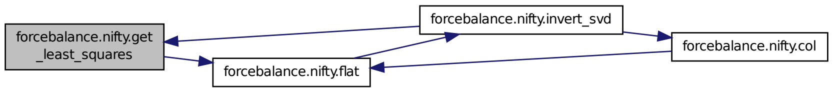
```
 __                  __
|                      |
| 1 (x0) (x0)^2 (x0)^3 |
| 1 (x1) (x1)^2 (x1)^3 |
| 1 (x2) (x2)^2 (x2)^3 |
| 1 (x3) (x3)^2 (x3)^3 |
| 1 (x4) (x4)^2 (x4)^3 |
|__                  __|
```

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 341 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.get _least_squares
forcebalance.nifty.invert_svd
forcebalance.nifty.col
forcebalance.nifty.flat

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

Invert a matrix using singular value decomposition.

Parameters

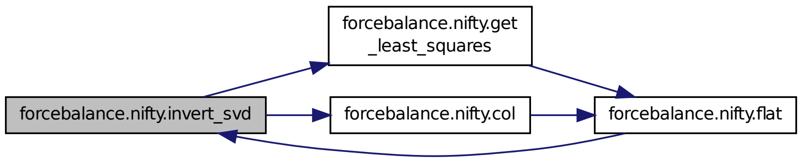
in	*X*	The matrix to be inverted
in	*thresh*	The SVD threshold; eigenvalues below this are not inverted but set to zero

Returns

Xt The inverted matrix

Definition at line 300 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.get _least_squares
forcebalance.nifty.invert_svd
forcebalance.nifty.col
forcebalance.nifty.flat

5.15.2.7 def forcebalance.nifty.isdecimal (*word*)

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

Parameters

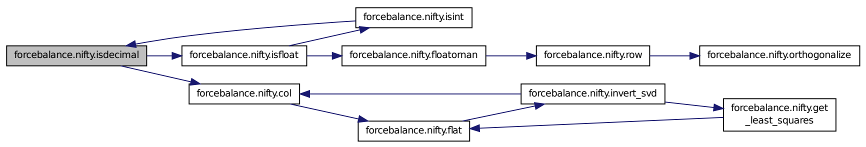
in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

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

Definition at line 226 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.isint
forcebalance.nifty.isdecimal
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize
forcebalance.nifty.col
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.flat

5.15.2.8 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

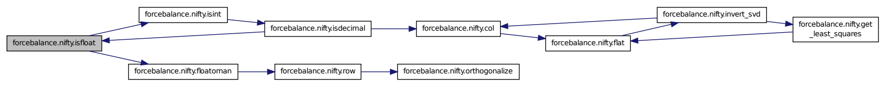
in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

answer Boolean which specifies whether the string is any number

Definition at line 216 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.isint
forcebalance.nifty.isfloat
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.flat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

5.15.2.9 def forcebalance.nifty.isint (*word*)

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

Parameters

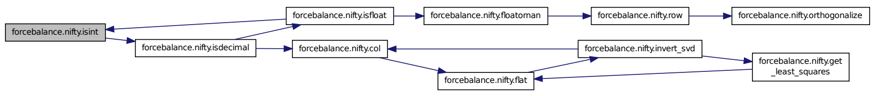
in	*word*	String (for instance, '123', '153.0', '2.', '-354')

Returns

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

Definition at line 205 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.isint
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.flat

5.15.2.10 def forcebalance.nifty.lp_dump (*obj,* *file,* *protocol =* `None`)

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

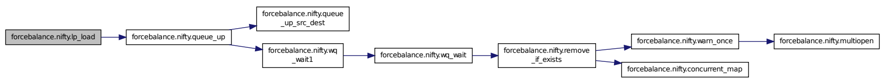
Definition at line 433 of file nifty.py.

5.15.2.11 def forcebalance.nifty.lp_load (*file*)

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

Definition at line 438 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.lp_load
forcebalance.nifty.queue_up
forcebalance.nifty.queue_up_src_dest
forcebalance.nifty.wq_wait1
forcebalance.nifty.wq_wait
forcebalance.nifty.remove_if_exists
forcebalance.nifty.warn_once
forcebalance.nifty.multiopen
forcebalance.nifty.concurrent_map

5.15.2.12 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 815 of file nifty.py.

5.15.2.13 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 287 of file nifty.py.

5.15.2.14 def forcebalance.nifty.pmat2d (*mat2d*, *precision* = `1`)

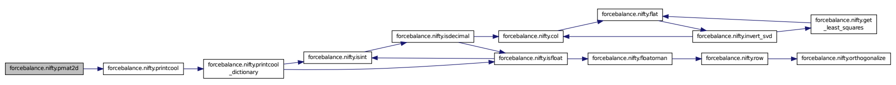
Printout of a 2-D matrix.

Parameters

`in`	*mat2d*	a 2-D matrix

Definition at line 60 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.pmat2d
forcebalance.nifty.printcool
forcebalance.nifty.printcool_dictionary
forcebalance.nifty.isint
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.flat
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

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

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:
		`#################`
		`### I am cool ###`
		`#################`

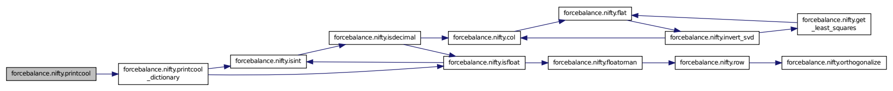
`in`	*sym*	The surrounding symbol
`in`	*bold*	Whether to use bold print
`in`	*color*	The ANSI color: 1 red 2 green 3 yellow 4 blue 5 magenta 6 cyan 7 white
`in`	*bottom*	The symbol for the bottom bar
`in`	*minwidth*	The minimum width for the box, if the text is very short then we insert the appropriate 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 149 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.printcool
forcebalance.nifty.printcool_dictionary
forcebalance.nifty.isint
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.flat
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

5.15.2.16 def forcebalance.nifty.printcool_dictionary (*Dict*, *title* = `"General options"`, *bold* = `False`, *color* = `2`, *keywidth* = `25`, *topwidth* = `50`)

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

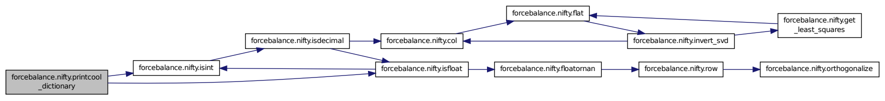
```
The keys in the dictionary are sorted before printing out.
```

Parameters

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

Definition at line 181 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.printcool_dictionary
forcebalance.nifty.isint
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.flat
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

5.15.2.17 def forcebalance.nifty.pvec1d (*vec1d*, *precision* = `1`)

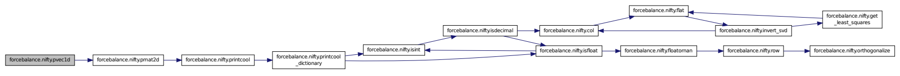
Printout of a 1-D vector.

Parameters

`in`	*vec1d*	a 1-D vector

Definition at line 49 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.pvec1d
forcebalance.nifty.pmat2d
forcebalance.nifty.printcool
forcebalance.nifty.printcool_dictionary
forcebalance.nifty.isint
forcebalance.nifty.isdecimal
forcebalance.nifty.col
forcebalance.nifty.flat
forcebalance.nifty.invert_svd
forcebalance.nifty.get_least_squares
forcebalance.nifty.isfloat
forcebalance.nifty.floatornan
forcebalance.nifty.row
forcebalance.nifty.orthogonalize

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

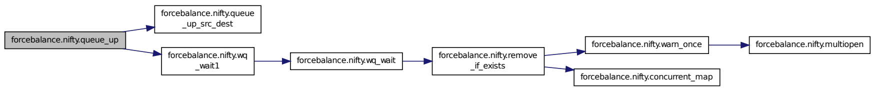
Submit a job to the Work Queue.

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 479 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.queue_up
forcebalance.nifty.queue_up_src_dest
forcebalance.nifty.wq_wait1
forcebalance.nifty.wq_wait
forcebalance.nifty.remove_if_exists
forcebalance.nifty.warn_once
forcebalance.nifty.multiopen
forcebalance.nifty.concurrent_map

5.15.2.19 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.

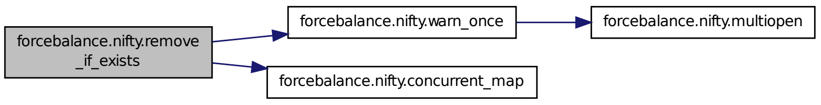
Definition at line 511 of file nifty.py.

5.15.2.20 def forcebalance.nifty.remove_if_exists (*fnm*)

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

Definition at line 666 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.remove_if_exists
forcebalance.nifty.warn_once
forcebalance.nifty.multiopen
forcebalance.nifty.concurrent_map

5.15.2.21 def forcebalance.nifty.row (*vec*)

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

Parameters

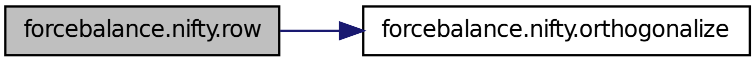
in	*vec*	The input vector that is to be made into a row

Returns

answer A row matrix

Definition at line 264 of file nifty.py.

Here is the call graph for this function:

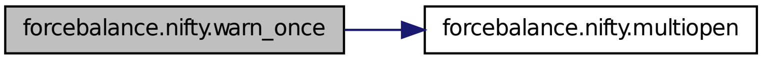


5.15.2.22 def forcebalance.nifty.warn_once (*warning*, *warnhash* = `None`)

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

Definition at line 761 of file nifty.py.

Here is the call graph for this function:

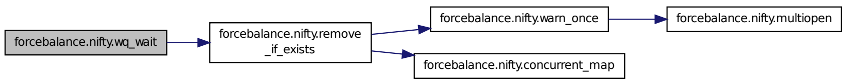


5.15.2.23 def forcebalance.nifty.wq_wait (*wq*, *verbose* = `False`)

This function waits until the work queue is completely empty.

Definition at line 582 of file nifty.py.

Here is the call graph for this function:

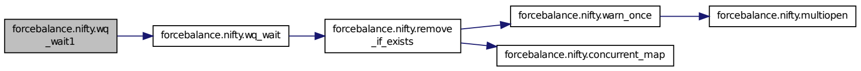

forcebalance.nifty.wq_wait
forcebalance.nifty.remove_if_exists
forcebalance.nifty.warn_once
forcebalance.nifty.multiopen
forcebalance.nifty.concurrent_map

5.15.2.24 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 532 of file nifty.py.

Here is the call graph for this function:


forcebalance.nifty.wq_wait1
forcebalance.nifty.wq_wait
forcebalance.nifty.remove_if_exists
forcebalance.nifty.warn_once
forcebalance.nifty.multiopen
forcebalance.nifty.concurrent_map

5.15.3 Variable Documentation

5.15.3.1 list forcebalance.nifty.specific_lst

Initial value:

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

Definition at line 610 of file nifty.py.

5.15.3.2 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 379 of file nifty.py.

5.16 forcebalance.objective Namespace Reference

ForceBalance objective function.

Classes

- class Objective

  Objective function.
- class Penalty

  Penalty functions for regularizing the force field optimizer.

Variables

- list Letters = ['X','G','H']

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

5.16.1 Detailed Description

ForceBalance objective function.

5.16.2 Variable Documentation

5.16.2.1 list forcebalance.objective.Letters = ['X','G','H']

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

Definition at line 18 of file objective.py.

5.17 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 **CopyAmoebaBondParameters**
- def **CopyAmoebaOutOfPlaneBendParameters**
- def **CopyAmoebaAngleParameters**
- def **CopyAmoebaInPlaneAngleParameters**
- def **CopyAmoebaVdwParameters**
- def **CopyAmoebaMultipoleParameters**
- def **CopyHarmonicBondParameters**
- def **CopyHarmonicAngleParameters**
- def **CopyPeriodicTorsionParameters**
- def **CopyNonbondedParameters**
- def **do_nothing**
- def **CopySystemParameters**
- def **UpdateSimulationParameters**

Variables

- dictionary suffix_dict

  Dictionary for building parameter identifiers.
- string pdict = "XML_Override"

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

5.17.1 Detailed Description

OpenMM input/output.

Author

Lee-Ping Wang

Date

04/2012

5.17.2 Variable Documentation

5.17.2.1 string forcebalance.openmmio.pdict = "XML_Override"

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

Definition at line 140 of file openmmio.py.

5.17.2.2 dictionary forcebalance.openmmio.suffix_dict

Initial value:

```
= { "HarmonicBondForce" : {"Bond" : ["class1","class2"]},
                 "HarmonicAngleForce" : {"Angle" : ["class1","class2","class3"],},
                 "PeriodicTorsionForce" : {"Proper" : ["class1","class2","class3","class4"],},
                 "NonbondedForce" : {"Atom": ["type"]},
                 "AmoebaBondForce" : {"Bond" : ["class1","class2"]},
                 "AmoebaAngleForce" : {"Angle" : ["class1","class2","class3"]},
                 "AmoebaStretchBendForce" : {"StretchBend" : ["class1","class2","class3"]},
                 "AmoebaVdwForce" : {"Vdw" : ["class"]},
                 "AmoebaMultipoleForce" : {"Multipole" : ["type","kz","kx"], "Polarize" : ["type"]},
                  "AmoebaUreyBradleyForce" : {"UreyBradley" : ["class1","class2","class3"]},
                  "Residues.Residue" : {"VirtualSite" : ["index"]}
                  }
```

Dictionary for building parameter identifiers.

As usual they go like this: Bond/length/OW.HW The dictionary is two-layered because the same interaction type (Bond) could be under two different parent types (HarmonicBondForce, AmoebaHarmonicBondForce)

Definition at line 126 of file openmmio.py.

5.18 forcebalance.optimizer Namespace Reference

Optimization algorithms.

Classes

- class Optimizer

  Optimizer class.

Functions

- def **Counter**
- def **GoodStep**

Variables

- int **ITERATION_NUMBER** = 0
- int **GOODSTEP** = 0

5.18.1 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

5.19 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.

5.19.1 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

5.19.2 Function Documentation

5.19.2.1 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 \em $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 363 of file parser.py.

5.19.2.2 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

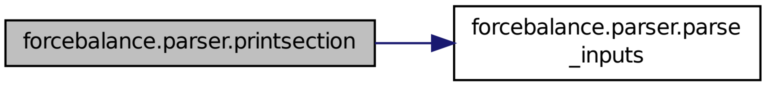
in	*heading*	Heading, either $options or $target
in	*optdict*	Options dictionary or None.
in	*typedict*	Option type dictionary, either gen_opts_types or tgt_opts_types specified in this file.

Returns

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

Definition at line 266 of file parser.py.

Here is the call graph for this function:


forcebalance.parser.printsection
forcebalance.parser.parse_inputs

5.19.3 Variable Documentation

5.19.3.1 dictionary forcebalance.parser.bkwd = {"simtype" : "type"}

Option maps for maintaining backward compatibility.

Definition at line 210 of file parser.py.

5.19.3.2 dictionary forcebalance.parser.gen_opts_defaults = {}

Default general options - basically a collapsed veresion of gen_opts_types.

Definition at line 194 of file parser.py.

5.19.3.3 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.

5.19.3.4 list forcebalance.parser.mainsections = [”SIMULATION”,”TARGET”,”OPTIONS”,”END”,”NONE”]

Listing of sections in the input file.

Definition at line 213 of file parser.py.

5.19.3.5 dictionary forcebalance.parser.ParsTab

Initial value:

```
= {"read_mvals" : read_mvals,
               "read_pvals" : read_pvals,
               "priors"     : read_priors,
               "internal"   : read_internals
               }
```

ParsTab that refers to subsection parsers.

Definition at line 244 of file parser.py.

5.19.3.6 dictionary forcebalance.parser.tgt_opts_defaults = {}

Default target options - basically a collapsed version of tgt_opts_types.

Definition at line 202 of file parser.py.

5.19.3.7 dictionary forcebalance.parser.tgt_opts_types

Default fitting target options.

Definition at line 123 of file parser.py.

5.20 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.

5.20.1 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

5.21 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.

5.21.1 Detailed Description

Q-Chem input file parser.

5.21.2 Variable Documentation

5.21.2.1 dictionary forcebalance.qchemio.pdict

Initial value:

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

Section -> Interaction type dictionary.

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

Definition at line 20 of file qchemio.py.

5.22 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**

5.22.1 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

5.22.2 Function Documentation

5.22.2.1 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.

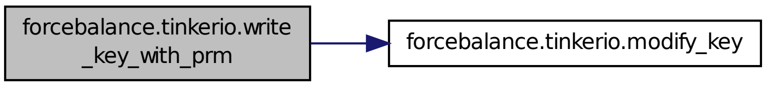
Definition at line 200 of file tinkerio.py.

5.22.2.2 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 144 of file tinkerio.py.

Here is the call graph for this function:


forcebalance.tinkerio.write_key_with_prm
forcebalance.tinkerio.modify_key

5.22.3 Variable Documentation

5.22.3.1 dictionary forcebalance.tinkerio.pdict

Initial value:

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

Definition at line 32 of file tinkerio.py.

5.23 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).

5.23.1 Detailed Description

Vibrational mode fitting module.

Author

Lee-Ping Wang

Date

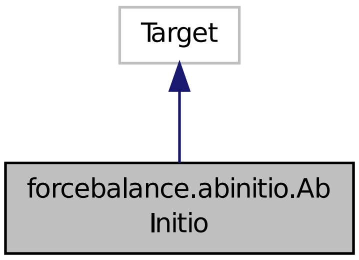
08/2012

6 Class Documentation

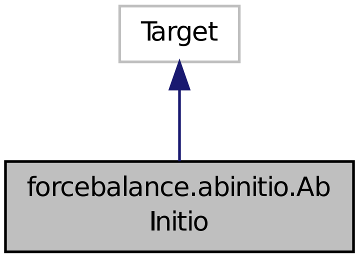
6.1 forcebalance.abinitio.AbInitio Class Reference

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_no_covariance_

  LPW 05-30-2012.
- def get_energy_force_covariance_

  LPW 01-11-2012.
- def get_resp_

  Electrostatic potential fitting.
- def **get_energy_force_**
- def **get**

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)
- esp_err

  Qualitative Indicator: "relative RMS" for electrostatic potential.
- **nf_err**
- **tq_err**
- 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**
- **covariance**
- **w_resp**
- **invdists**
- **respterm**
- **objective**

6.1.1 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.

6.1.2 Constructor & Destructor Documentation

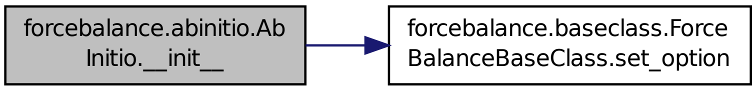
6.1.2.1 def forcebalance.abinitio.AbInitio.__init__ (*self*, *options*, *tgt_opts*, *forcefield*)

Initialization; define a few core concepts.

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

Definition at line 54 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.AbInitio.__init__
forcebalance.baseclass.ForceBalanceBaseClass.set_option

6.1.3 Member Function Documentation

6.1.3.1 def forcebalance.abinitio.AbInitio.get_energy_force_covariance_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

LPW 01-11-2012.

```
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.

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 and fitting simulations 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

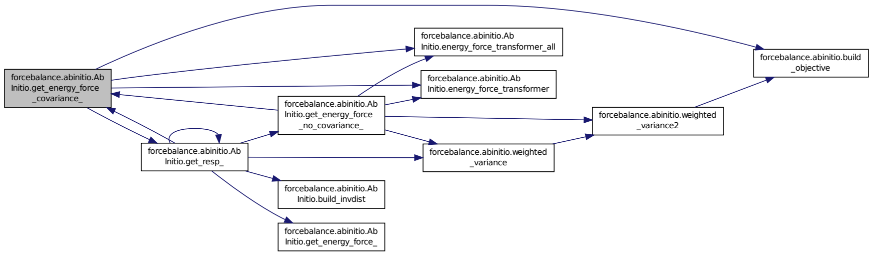
in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 888 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.Ab Initio.get_energy_force _covariance_
forcebalance.abinitio.Ab Initio.energy_force_transformer_all
forcebalance.abinitio.Ab Initio.energy_force_transformer
forcebalance.abinitio.build _objective
forcebalance.abinitio.Ab Initio.get_energy_force _no_covariance_
forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.Ab Initio.get_resp_
forcebalance.abinitio.weighted _variance
forcebalance.abinitio.Ab Initio.build_invdist
forcebalance.abinitio.Ab Initio.get_energy_force_

6.1.3.2 def forcebalance.abinitio.AbInitio.get_energy_force_no_covariance_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

LPW 05-30-2012.

```
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: Removed because it adds a factor of
NCP1 to the memory requirement.  Instead we're trying
Gauss-Newton approximation to the Hessian.

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
```

Todo Parallelization over snapshots is not implemented yet

Parameters

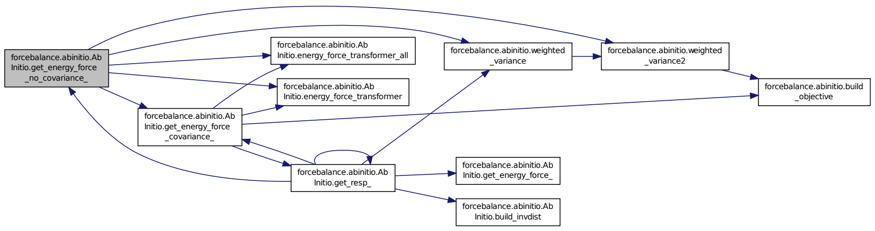
in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 507 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.Ab Initio.get_energy_force _no_covariance_
forcebalance.abinitio.Ab Initio.energy_force_transformer_all
forcebalance.abinitio.weighted _variance
forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.Ab Initio.energy_force_transformer
forcebalance.abinitio.build _objective
forcebalance.abinitio.Ab Initio.get_energy_force _covariance_
forcebalance.abinitio.Ab Initio.get_resp_
forcebalance.abinitio.Ab Initio.get_energy_force_
forcebalance.abinitio.Ab Initio.build_invdist

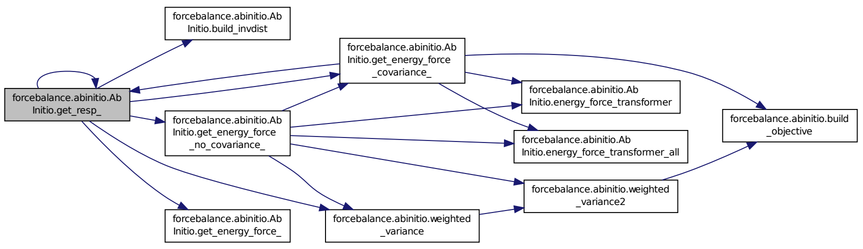
6.1.3.3 def forcebalance.abinitio.AbInitio.get_resp_ (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

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 1096 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.Ab Initio.build_invdist
forcebalance.abinitio.Ab Initio.get_energy_force _covariance_
forcebalance.abinitio.Ab Initio.get_resp_
forcebalance.abinitio.Ab Initio.energy_force_transformer
forcebalance.abinitio.build _objective
forcebalance.abinitio.Ab Initio.get_energy_force _no_covariance_
forcebalance.abinitio.Ab Initio.energy_force_transformer_all
forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.Ab Initio.get_energy_force_
forcebalance.abinitio.weighted _variance

6.1.3.4 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)

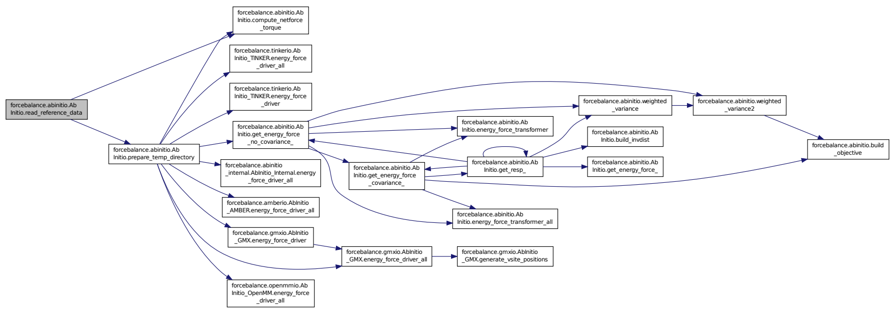
```
After reading in the information from qdata.txt, it is converted
into the GROMACS energy units (kind of an arbitrary choice);
forces (kind of a misnomer in qdata.txt) are multipled by -1
to convert gradients to forces.

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 311 of file abinitio.py.

Here is the call graph for this function:


forcebalance.abinitio.Ab Initio.read_reference_data
forcebalance.abinitio.Ab Initio.prepare_temp_directory
forcebalance.abinitio.Ab Initio.compute_netforce _torque
forcebalance.tinkerio.Ab Initio_TINKER.energy_force _driver_all
forcebalance.tinkerio.Ab Initio_TINKER.energy_force _driver
forcebalance.abinitio.Ab Initio.get_energy_force _no_covariance_
forcebalance.abinitio _internal.AbInitio_Internal.energy _force_driver_all
forcebalance.amberio.AbInitio _AMBER.energy_force_driver_all
forcebalance.gmxio.AbInitio _GMX.energy_force_driver
forcebalance.gmxio.AbInitio _GMX.energy_force_driver_all
forcebalance.gmxio.AbInitio _GMX.generate_vsite_positions
forcebalance.openmmio.Ab Initio_OpenMM.energy_force _driver_all
forcebalance.abinitio.Ab Initio.get_energy_force _covariance_
forcebalance.abinitio.Ab Initio.energy_force_transformer
forcebalance.abinitio.Ab Initio.get_resp_
forcebalance.abinitio.Ab Initio.energy_force_transformer_all
forcebalance.abinitio.weighted _variance
forcebalance.abinitio.weighted _variance2
forcebalance.abinitio.Ab Initio.build_invdist
forcebalance.abinitio.Ab Initio.get_energy_force_
forcebalance.abinitio.build _objective

6.1.4 Member Data Documentation

6.1.4.1 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 143 of file abinitio.py.

6.1.4.2 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 152 of file abinitio.py.

6.1.4.3 forcebalance.abinitio.AbInitio.save_vmvals

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

Definition at line 154 of file abinitio.py.

6.1.4.4 forcebalance.abinitio.AbInitio.use_nft

Whether to compute net forces and torques, or not.

Definition at line 132 of file abinitio.py.

6.1.4.5 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. 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 105 of file abinitio.py.

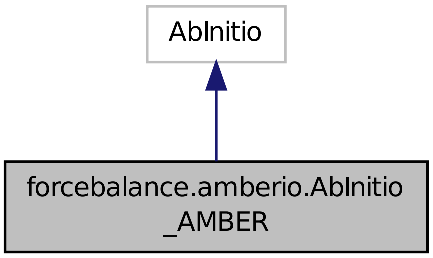
The documentation for this class was generated from the following file:

- abinitio.py

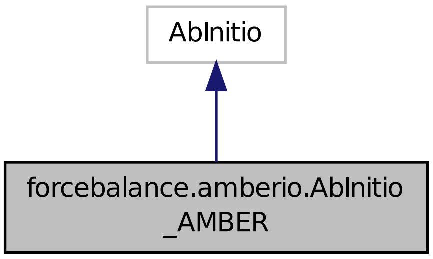
6.2 forcebalance.amberio.AbInitio_AMBER Class Reference

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__**
- def **prepare_temp_directory**
- 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.

6.2.1 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.

6.2.2 Member Data Documentation

6.2.2.1 forcebalance.amberio.AbInitio_AMBER.all_at_once

all_at_once is not implemented.

Definition at line 176 of file amberio.py.

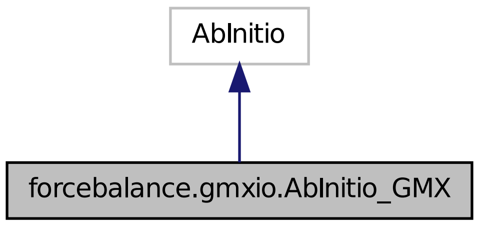
The documentation for this class was generated from the following file:

- amberio.py

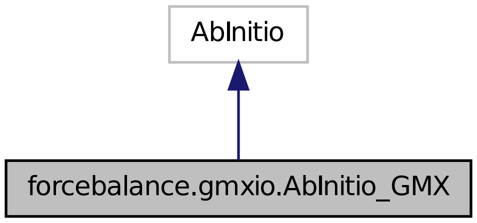
6.3 forcebalance.gmxio.AbInitio_GMX Class Reference

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__**
- def **read_topology**
- def **prepare_temp_directory**
- 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**
- **topology_flag**

6.3.1 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 369 of file gmxio.py.

6.3.2 Member Function Documentation

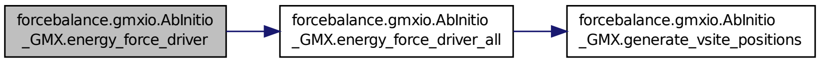
6.3.2.1 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver (*self*, *shot*)

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

This does not require GROMACS-X2.

Definition at line 439 of file gmxio.py.

Here is the call graph for this function:


forcebalance.gmxio.AbInitio _GMX.energy_force_driver
forcebalance.gmxio.AbInitio _GMX.energy_force_driver_all
forcebalance.gmxio.AbInitio _GMX.generate_vsite_positions

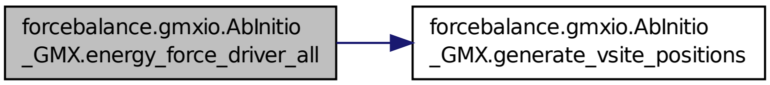
6.3.2.2 def forcebalance.gmxio.AbInitio_GMX.energy_force_driver_all (*self*)

Computes the energy and force using GROMACS for a trajectory.

This does not require GROMACS-X2.

Definition at line 466 of file gmxio.py.

Here is the call graph for this function:


forcebalance.gmxio.AbInitio _GMX.energy_force_driver_all
forcebalance.gmxio.AbInitio _GMX.generate_vsite_positions

6.3.2.3 def forcebalance.gmxio.AbInitio_GMX.generate_vsite_positions (*self*)

Call mdrun in order to update the virtual site positions.

Definition at line 493 of file gmxio.py.

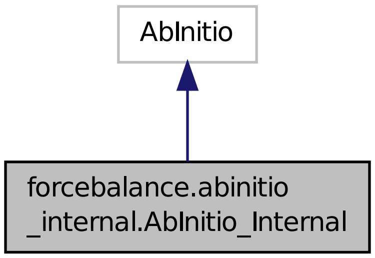
The documentation for this class was generated from the following file:

- gmxio.py

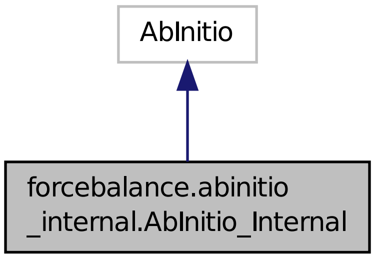
6.4 forcebalance.abinitio_internal.AbInitio_Internal Class Reference

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__**
- 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.

6.4.1 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.

6.4.2 Member Function Documentation

6.4.2.1 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.

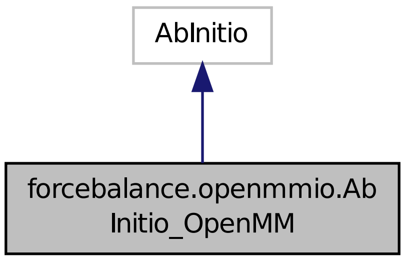
The documentation for this class was generated from the following file:

- abinitio_internal.py

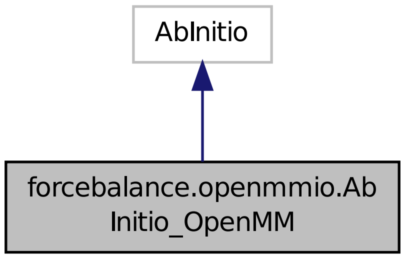
6.5 forcebalance.openmmio.AbInitio_OpenMM Class Reference

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__**
- def **read_topology**
- def **prepare_temp_directory**
- 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**

6.5.1 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 212 of file openmmio.py.

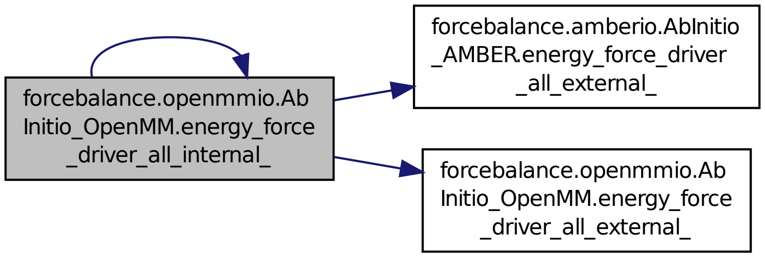
6.5.2 Member Function Documentation

6.5.2.1 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 291 of file openmmio.py.

Here is the call graph for this function:


forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_internal_
forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all_external_
forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all_external_

6.5.3 Member Data Documentation

6.5.3.1 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 224 of file openmmio.py.

6.5.3.2 forcebalance.openmmio.AbInitio_OpenMM.simulation

Create the simulation object within this class itself.

Definition at line 256 of file openmmio.py.

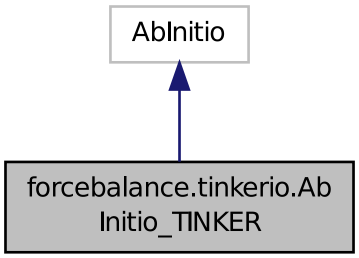
The documentation for this class was generated from the following file:

- openmmio.py

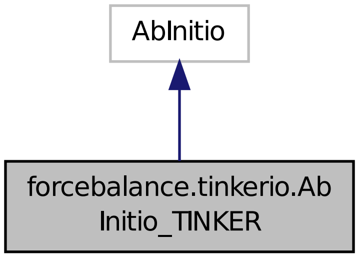
6.6 forcebalance.tinkerio.AbInitio_TINKER Class Reference

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__**
- def **prepare_temp_directory**
- 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.

6.6.1 Detailed Description

Subclass of Target for force and energy matching using TINKER.

Implements the prepare and energy_force_driver methods.

Definition at line 290 of file tinkerio.py.

6.6.2 Member Data Documentation

6.6.2.1 forcebalance.tinkerio.AbInitio_TINKER.all_at_once

all_at_once is not implemented.

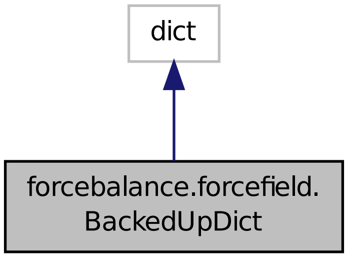
Definition at line 300 of file tinkerio.py.

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

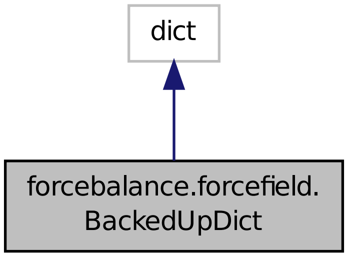
- tinkerio.py

6.7 forcebalance.forcefield.BackedUpDict Class Reference

Inheritance diagram for forcebalance.forcefield.BackedUpDict:



Collaboration diagram for forcebalance.forcefield.BackedUpDict:



Public Member Functions

- def **__init__**
- def **__missing__**

Public Attributes

- **backup_dict**

6.7.1 Detailed Description

Definition at line 174 of file forcefield.py.

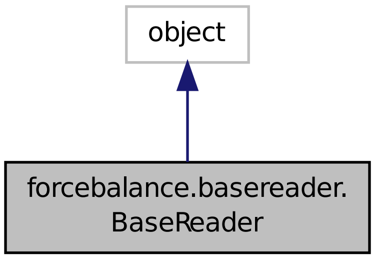
The documentation for this class was generated from the following file:

- forcefield.py

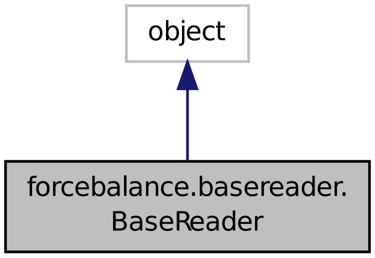
6.8 forcebalance.basereader.BaseReader Class Reference

The 'reader' class.

Inheritance diagram for forcebalance.basereader.BaseReader:



Collaboration 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**

6.8.1 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.

6.8.2 Member Function Documentation

6.8.2.1 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'

Definition at line 68 of file basereader.py.

6.8.3 Member Data Documentation

6.8.3.1 forcebalance.basereader.BaseReader.adict

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

Definition at line 33 of file basereader.py.

6.8.3.2 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.

Definition at line 38 of file basereader.py.

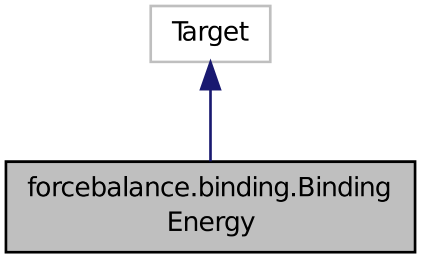
The documentation for this class was generated from the following file:

- basereader.py

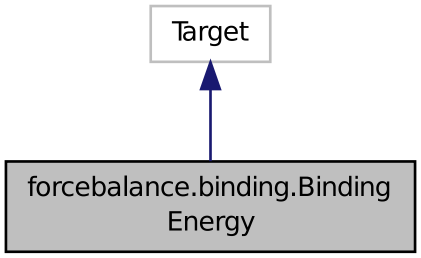
6.9 forcebalance.binding.BindingEnergy Class Reference

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__**
- def **indicate**
- def **get**

Public Attributes

- **inter_opts**
- **PrintDict**
- **RMSDDict**
- **rmsd_part**
- **energy_part**
- **objective**

6.9.1 Detailed Description

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

Definition at line 122 of file binding.py.

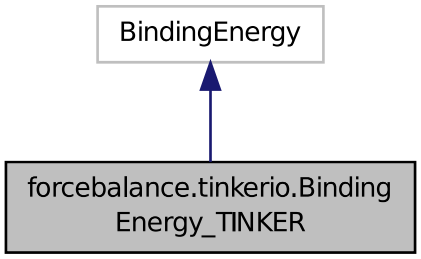
The documentation for this class was generated from the following file:

- binding.py

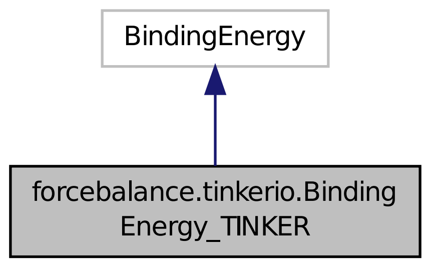
6.10 forcebalance.tinkerio.BindingEnergy_TINKER Class Reference

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__**
- def **prepare_temp_directory**
- def **system_driver**

Public Attributes

- **optprog**

6.10.1 Detailed Description

Subclass of BindingEnergy for binding energy matching using TINKER.

Definition at line 479 of file tinkerio.py.

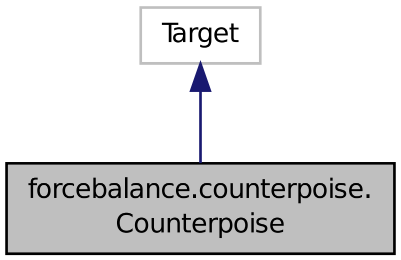
The documentation for this class was generated from the following file:

- tinkerio.py

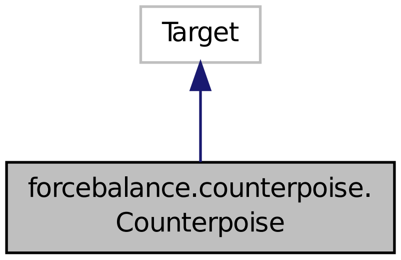
6.11 forcebalance.counterpoise.Counterpoise Class Reference

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**

6.11.1 Detailed Description

Target subclass for matching the counterpoise correction.

Definition at line 31 of file counterpoise.py.

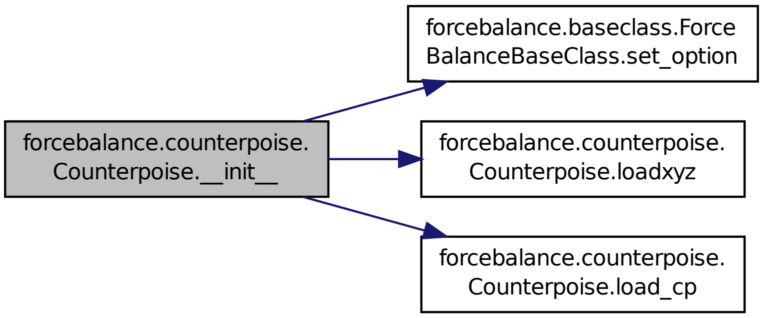
6.11.2 Constructor & Destructor Documentation

6.11.2.1 def forcebalance.counterpoise.Counterpoise.__init__ (*self*, *options*, *tgt_opts*, *forcefield*)

To instantiate Counterpoise, we read the coordinates and counterpoise data.

Definition at line 35 of file counterpoise.py.

Here is the call graph for this function:


forcebalance.counterpoise.Counterpoise.__init__
forcebalance.baseclass.ForceBalanceBaseClass.set_option
forcebalance.counterpoise.Counterpoise.loadxyz
forcebalance.counterpoise.Counterpoise.load_cp

6.11.3 Member Function Documentation

6.11.3.1 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

Definition at line 122 of file counterpoise.py.

6.11.3.2 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.

6.11.3.3 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.

6.11.4 Member Data Documentation

6.11.4.1 forcebalance.counterpoise.Counterpoise.xyzs

Number of snapshots.

XYZ elements and coordinates

Definition at line 49 of file counterpoise.py.

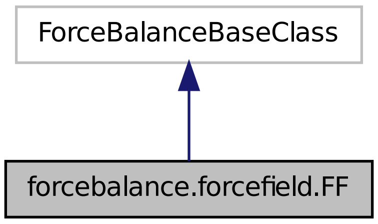
The documentation for this class was generated from the following file:

- counterpoise.py

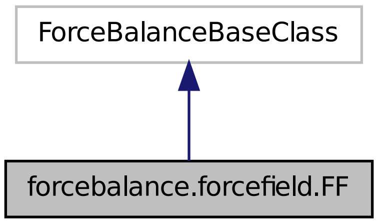
6.12 forcebalance.forcefield.FF Class Reference

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]].

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**

6.12.1 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 195 of file forcefield.py.

6.12.2 Constructor & Destructor Documentation

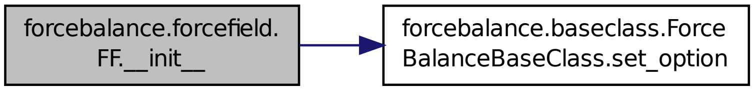
6.12.2.1 def forcebalance.forcefield.FF.__init__ (*self*, *options*, *verbose* = `True`)

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 203 of file forcefield.py.

Here is the call graph for this function:



6.12.3 Member Function Documentation

6.12.3.1 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,
<tt> ffname force_field.xml:openmm </tt> 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 <tt> parameter identifier -> index in parameter vector </tt>
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 <tt> unique parameter identifier <tt>.  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

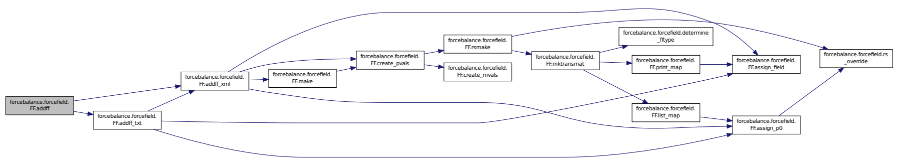
My program currently assumes that we are only using one MM program per job. If we use CHARMM and GROMACS to perform simulations as part of the same TARGET, we will get messed up. Maybe this needs to be fixed in the future, with program prefixes to parameters like C_ , G_ .. or simply unit conversions, 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 394 of file forcefield.py.

Here is the call graph for this function:


forcebalance.forcefield.FF.addff
forcebalance.forcefield.FF.addff_txt
forcebalance.forcefield.FF.addff_xml
forcebalance.forcefield.FF.make
forcebalance.forcefield.FF.create_pvals
forcebalance.forcefield.FF.rsmake
forcebalance.forcefield.FF.create_mvals
forcebalance.forcefield.FF.mktransmat
forcebalance.forcefield.determine_fftype
forcebalance.forcefield.FF.print_map
forcebalance.forcefield.FF.list_map
forcebalance.forcefield.FF.assign_field
forcebalance.forcefield.FF.assign_p0
forcebalance.forcefield.rs_override

6.12.3.2 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

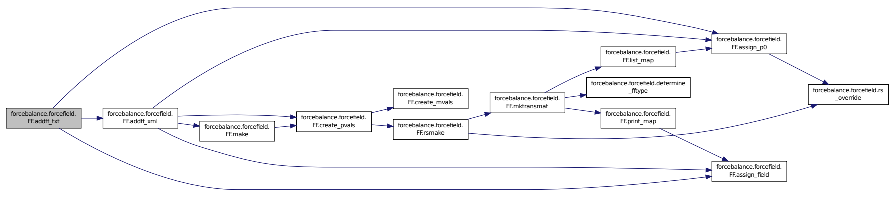
Additionally, when 'PARM' is encountered:

- 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 464 of file forcefield.py.

Here is the call graph for this function:


forcebalance.forcefield.FF.addff_txt
forcebalance.forcefield.FF.addff_xml
forcebalance.forcefield.FF.make
forcebalance.forcefield.FF.create_pvals
forcebalance.forcefield.FF.create_mvals
forcebalance.forcefield.FF.rsmake
forcebalance.forcefield.FF.mktransmat
forcebalance.forcefield.FF.list_map
forcebalance.forcefield.determine_fftype
forcebalance.forcefield.FF.print_map
forcebalance.forcefield.FF.assign_p0
forcebalance.forcefield.rs_override
forcebalance.forcefield.FF.assign_field

6.12.3.3 def forcebalance.forcefield.FF.addff_xml (*self, ffname*)

Parse an XML force field file and create important instance variables.

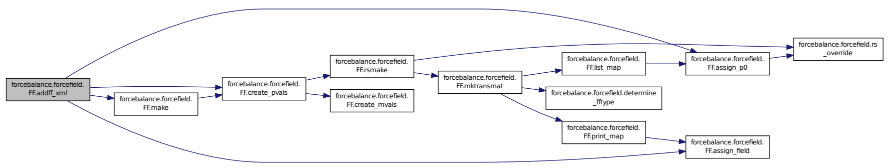
```
This was modeled after addff_txt, but XML and text files are
fundamentally different, necessitating two different methods.

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 572 of file forcefield.py.

Here is the call graph for this function:


forcebalance.forcefield.FF.addff_xml
forcebalance.forcefield.FF.make
forcebalance.forcefield.FF.create_pvals
forcebalance.forcefield.FF.rsmake
forcebalance.forcefield.FF.create_mvals
forcebalance.forcefield.FF.mktransmat
forcebalance.forcefield.FF.list_map
forcebalance.forcefield.determine_fftype
forcebalance.forcefield.FF.print_map
forcebalance.forcefield.FF.assign_p0
forcebalance.forcefield.FF.assign_field
forcebalance.forcefield.rs_override

6.12.3.4 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 1134 of file forcefield.py.

6.12.3.5 def forcebalance.forcefield.FF.assign_p0 (*self*, *idx*, *val*)

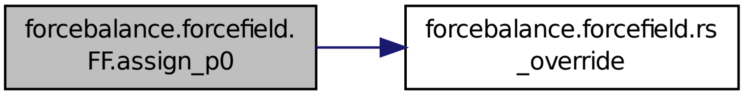
Assign physical parameter values to the 'pvals0' array.

Parameters

in	*idx*	The index to which we assign the parameter value.
in	*val*	The parameter value to be inserted.

Definition at line 1116 of file forcefield.py.

Here is the call graph for this function:



6.12.3.6 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 850 of file forcefield.py.

6.12.3.7 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

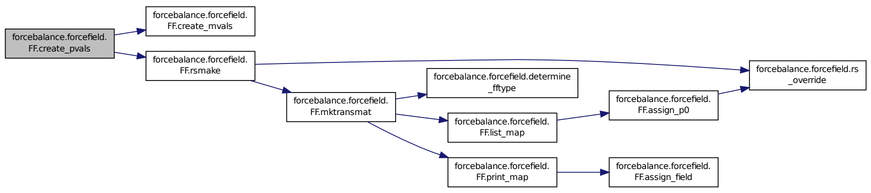
in	*mvals*	The mathematical parameters

Returns

pvals The physical parameters

Definition at line 814 of file forcefield.py.

Here is the call graph for this function:



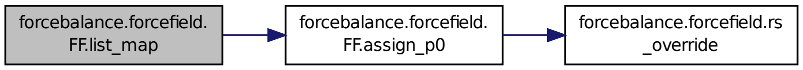
6.12.3.8 def forcebalance.forcefield.FF.list_map (*self*)

Create the plist, which is like a reversed version of the parameter map.

More convenient for printing.

Definition at line 1093 of file forcefield.py.

Here is the call graph for this function:



6.12.3.9 def forcebalance.forcefield.FF.make (*self*, *vals*, *use_pvals* = `False`, *printdir* = `None`, *precision* = `12`)

Create a new force field using provided parameter values.

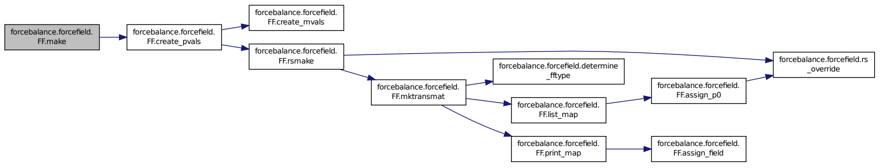
```
This big kahuna does a number of things:
1) Creates the physical parameters from the mathematical parameters
2) Creates force fields with physical parameters substituted in
3) Prints the force fields to the specified file.

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 620 of file forcefield.py.

Here is the call graph for this function:


forcebalance.forcefield. FF.make
forcebalance.forcefield. FF.create_pvals
forcebalance.forcefield. FF.create_mvals
forcebalance.forcefield. FF.rsmake
forcebalance.forcefield. FF.mktransmat
forcebalance.forcefield.determine _fftype
forcebalance.forcefield. FF.list_map
forcebalance.forcefield. FF.print_map
forcebalance.forcefield. FF.assign_p0
forcebalance.forcefield. FF.assign_field
forcebalance.forcefield.rs _override

6.12.3.10 def forcebalance.forcefield.FF.mktransmat (*self*)

Create the transformation matrix to rescale and rotate the mathematical parameters.

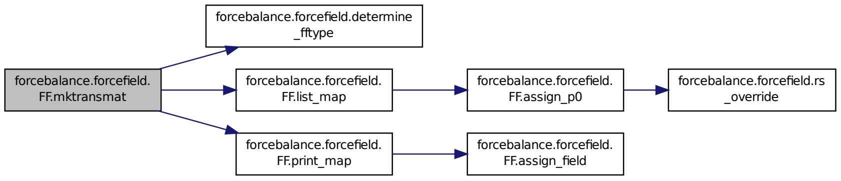
```
For point charge parameters, project out perturbations that
change the total charge.

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 943 of file forcefield.py.

Here is the call graph for this function:

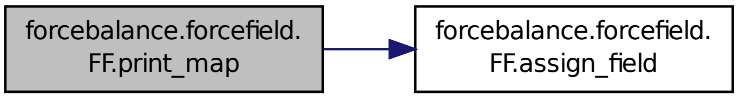

forcebalance.forcefield. FF.mktransmat
forcebalance.forcefield.determine _fftype
forcebalance.forcefield. FF.list_map
forcebalance.forcefield. FF.print_map
forcebalance.forcefield. FF.assign_p0
forcebalance.forcefield. FF.assign_field
forcebalance.forcefield.rs _override

6.12.3.11 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 1105 of file forcefield.py.

Here is the call graph for this function:



6.12.3.12 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.
```

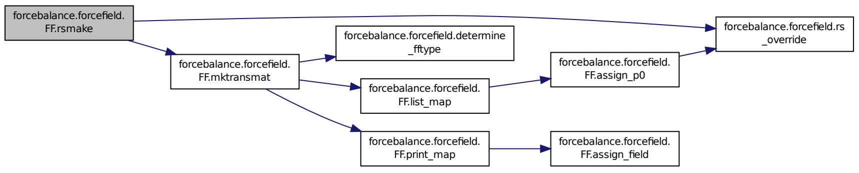
Todo Pass in rsfactors through the input file

Parameters

`in`	*printfacs*	List for printing out the resecaling factors

Definition at line 867 of file forcefield.py.

Here is the call graph for this function:



6.12.4 Member Data Documentation

6.12.4.1 forcebalance.forcefield.FF.excision

Indices to exclude from optimization / Hessian inversion.

Some customized constraints here.

Quadrupoles must be traceless

Definition at line 256 of file forcefield.py.

6.12.4.2 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 236 of file forcefield.py.

6.12.4.3 forcebalance.forcefield.FF.linedestroy_save

Destruction dictionary (experimental).

Definition at line 315 of file forcefield.py.

6.12.4.4 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 248 of file forcefield.py.

6.12.4.5 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 313 of file forcefield.py.

6.12.4.6 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 250 of file forcefield.py.

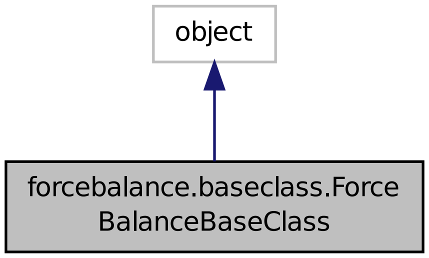
The documentation for this class was generated from the following file:

- forcefield.py

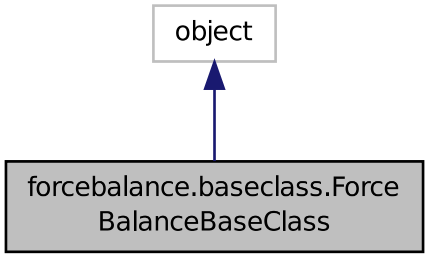
6.13 forcebalance.baseclass.ForceBalanceBaseClass Class Reference

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

Inheritance diagram for forcebalance.baseclass.ForceBalanceBaseClass:



Collaboration diagram for forcebalance.baseclass.ForceBalanceBaseClass:



Public Member Functions

- def **__init__**
- def **set_option**

Public Attributes

- **PrintOptionDict**
- **verbose_options**

6.13.1 Detailed Description

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

Definition at line 6 of file baseclass.py.

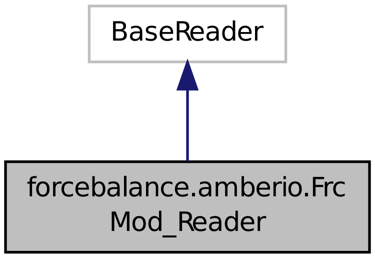
The documentation for this class was generated from the following file:

- baseclass.py

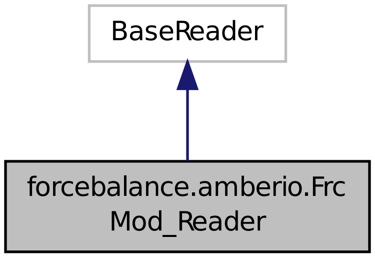
6.14 forcebalance.amberio.FrcMod_Reader Class Reference

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**

6.14.1 Detailed Description

Finite state machine for parsing FrcMod force field file.

Definition at line 96 of file amberio.py.

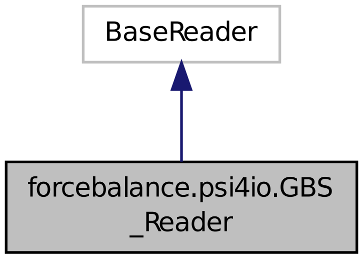
The documentation for this class was generated from the following file:

- amberio.py

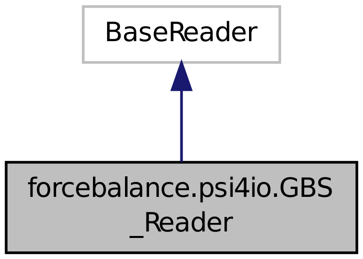
6.15 forcebalance.psi4io.GBS_Reader Class Reference

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**
- def feed

  Feed in a line.

Public Attributes

- **element**
- **amom**
- **last_amom**
- **basis_number**
- **contraction_number**
- **adict**
- **isdata**
- **destroy**

6.15.1 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.

6.15.2 Member Function Documentation

6.15.2.1 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.

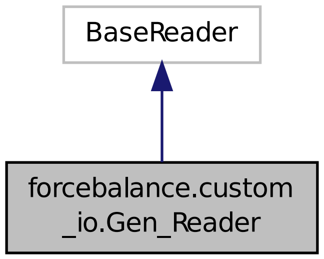
The documentation for this class was generated from the following file:

- psi4io.py

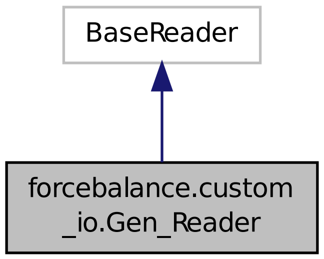
6.16 forcebalance.custom_io.Gen_Reader Class Reference

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**

6.16.1 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.

6.16.2 Member Function Documentation

6.16.2.1 def forcebalance.custom_io.Gen_Reader.feed (*self, line*)

Feed in a line.

Parameters

in	*line*	The line of data

Definition at line 57 of file custom_io.py.

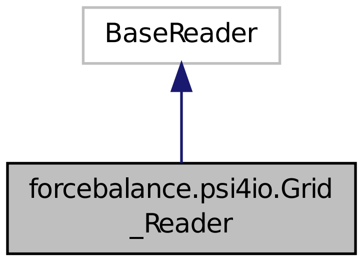
The documentation for this class was generated from the following file:

- custom_io.py

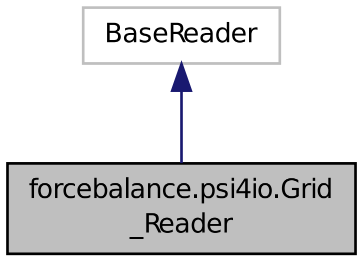
6.17 forcebalance.psi4io.Grid_Reader Class Reference

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**
- def feed

  Feed in a line.

Public Attributes

- **element**
- **point**
- **radii**
- **isdata**

6.17.1 Detailed Description

Finite state machine for parsing DVR grid files.

Definition at line 246 of file psi4io.py.

6.17.2 Member Function Documentation

6.17.2.1 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.

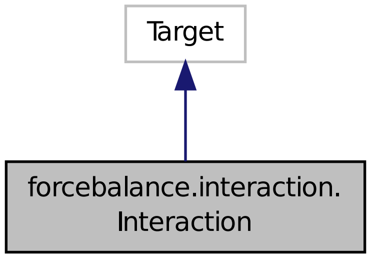
The documentation for this class was generated from the following file:

- psi4io.py

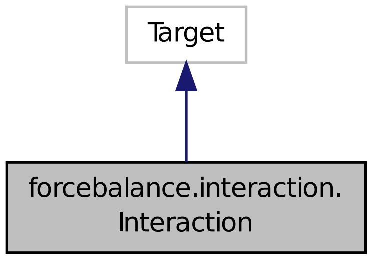
6.18 forcebalance.interaction.Interaction Class Reference

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__**
- 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**

6.18.1 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.

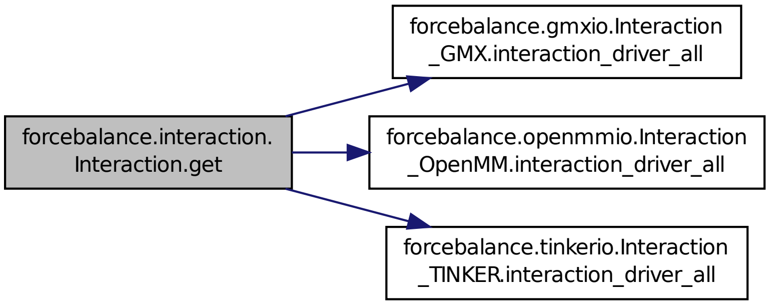
6.18.2 Member Function Documentation

6.18.2.1 def forcebalance.interaction.Interaction.get (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

Definition at line 160 of file interaction.py.

Here is the call graph for this function:


forcebalance.interaction.Interaction.get
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all

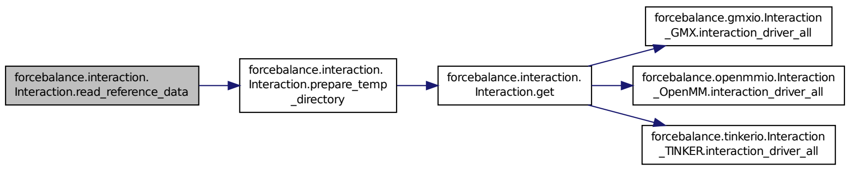
6.18.2.2 def forcebalance.interaction.Interaction.read_reference_data (*self*)

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:


forcebalance.interaction.
Interaction.read_reference_data
forcebalance.interaction.
Interaction.prepare_temp
_directory
forcebalance.interaction.
Interaction.get
forcebalance.gmxio.Interaction
_GMX.interaction_driver_all
forcebalance.openmmio.Interaction
_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction
_TINKER.interaction_driver_all

6.18.3 Member Data Documentation

6.18.3.1 forcebalance.interaction.Interaction.divisor

Read in the reference data.

Prepare the temporary directory

Definition at line 93 of file interaction.py.

6.18.3.2 forcebalance.interaction.Interaction.eqm

Set upper cutoff energy.

Reference (QM) interaction energies

Definition at line 69 of file interaction.py.

6.18.3.3 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 57 of file interaction.py.

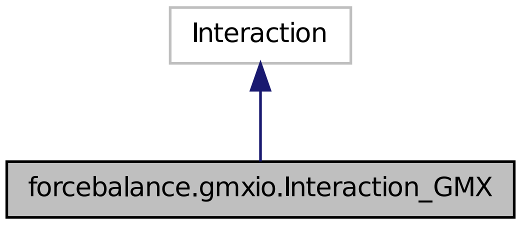
The documentation for this class was generated from the following file:

- interaction.py

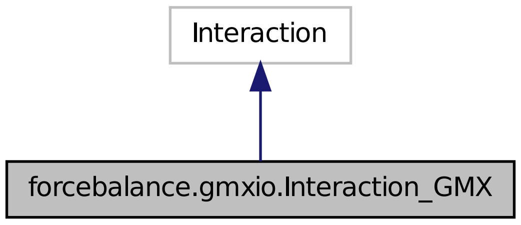
6.19 forcebalance.gmxio.Interaction_GMX Class Reference

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__**
- def **prepare_temp_directory**
- 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**

6.19.1 Detailed Description

Subclass of Interaction for interaction energy matching using GROMACS.

Definition at line 506 of file gmxio.py.

6.19.2 Member Function Documentation

6.19.2.1 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 539 of file gmxio.py.

6.19.2.2 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 544 of file gmxio.py.

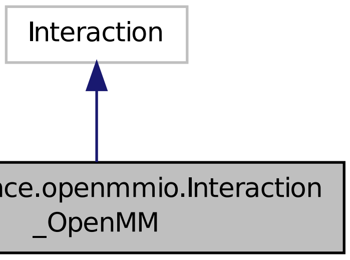
The documentation for this class was generated from the following file:

- gmxio.py

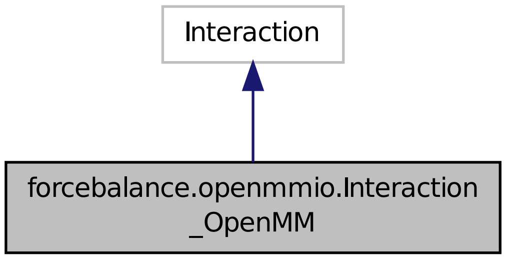
6.20 forcebalance.openmmio.Interaction_OpenMM Class Reference

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__**
- def **prepare_temp_directory**
- 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.

6.20.1 Detailed Description

Subclass of Target for interaction matching using OpenMM.

Definition at line 353 of file openmmio.py.

6.20.2 Member Data Documentation

6.20.2.1 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 382 of file openmmio.py.

6.20.2.2 forcebalance.openmmio.Interaction_OpenMM.trajfnm

Name of the trajectory file containing snapshots.

Definition at line 358 of file openmmio.py.

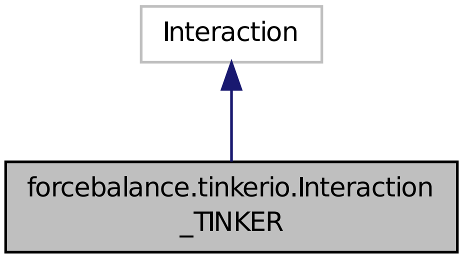
The documentation for this class was generated from the following file:

- openmmio.py

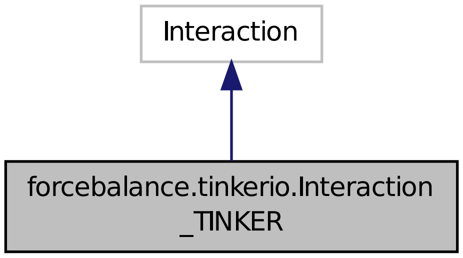
6.21 forcebalance.tinkerio.Interaction_TINKER Class Reference

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__**

- def **prepare_temp_directory**
- def **energy_driver_all**
- def **interaction_driver_all**

Public Attributes

- trajfnm

  Name of the trajectory.

6.21.1 Detailed Description

Subclass of Target for interaction matching using TINKER.

Definition at line 578 of file tinkerio.py.

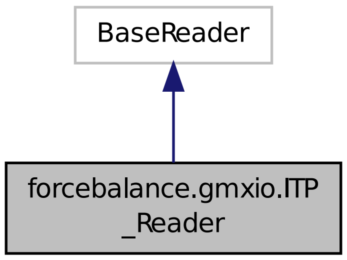
The documentation for this class was generated from the following file:

- tinkerio.py

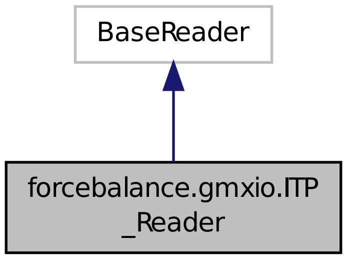
6.22 forcebalance.gmxio.ITP_Reader Class Reference

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**

6.22.1 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 214 of file gmxio.py.

6.22.2 Member Function Documentation

6.22.2.1 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.

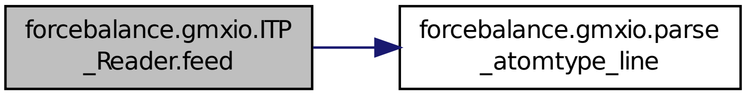
The section is essentially a case statement that picks out the appropriate interaction type and makes a list of the atoms involved

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

Definition at line 255 of file gmxio.py.

Here is the call graph for this function:


forcebalance.gmxio.ITP _Reader.feed
forcebalance.gmxio.parse _atomtype_line

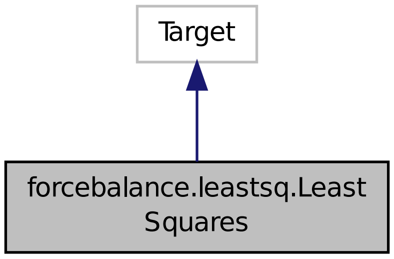
The documentation for this class was generated from the following file:

- gmxio.py

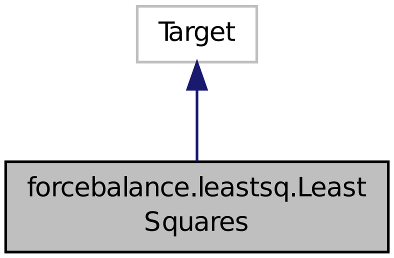
6.23 forcebalance.leastsq.LeastSquares Class Reference

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__**
- def **indicate**
- def get

  LPW 05-30-2012.

Public Attributes

- call_derivatives

  Number of snapshots.
- MAQ

  Dictionary for derivative terms.
- **D**
- **objective**

6.23.1 Detailed Description

Subclass of Target for general least squares fitting.

Definition at line 32 of file leastsq.py.

6.23.2 Member Function Documentation

6.23.2.1 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

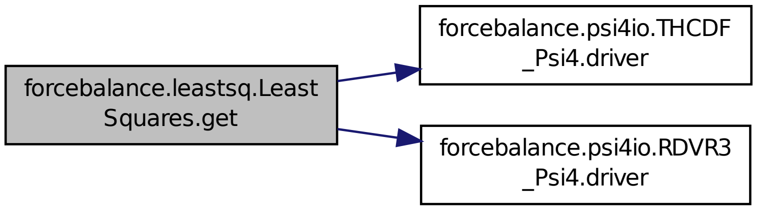
in	*mvals*	Mathematical parameter values
in	*AGrad*	Switch to turn on analytic gradient
in	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 72 of file leastsq.py.

Here is the call graph for this function:


forcebalance.leastsq.Least Squares.get
forcebalance.psi4io.THCDF _Psi4.driver
forcebalance.psi4io.RDVR3 _Psi4.driver

6.23.3 Member Data Documentation

6.23.3.1 forcebalance.leastsq.LeastSquares.call_derivatives

Number of snapshots.

Which parameters are differentiated?

Definition at line 48 of file leastsq.py.

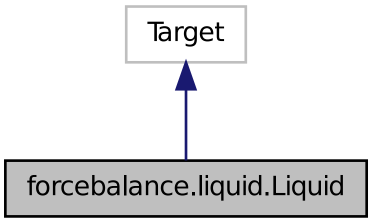
The documentation for this class was generated from the following file:

- leastsq.py

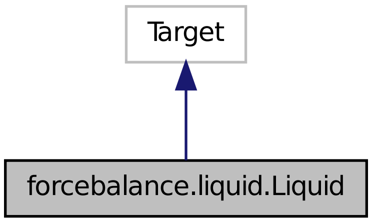
6.24 forcebalance.liquid.Liquid Class Reference

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 **indicate**
- def **objective_term**
- def **submit_jobs**
- def get

  Fitting of liquid bulk properties.

Public Attributes

- SavedMVal

  Read the reference data.
- SavedTraj

Saved trajectories for all iterations and all temperatures :)

- MBarEnergy

  Evaluated energies for all trajectories (i.e.
- **RefData**
- **PhasePoints**
- **Labels**
- w_rho

  Density.
- **w_hvap**
- **w_alpha**
- **w_kappa**
- **w_cp**
- **w_eps0**

6.24.1 Detailed Description

Subclass of Target for liquid property matching.

Definition at line 45 of file liquid.py.

6.24.2 Constructor & Destructor Documentation

6.24.2.1 def forcebalance.liquid.Liquid.__init__ (*self*, *options*, *tgt_opts*, *forcefield*)

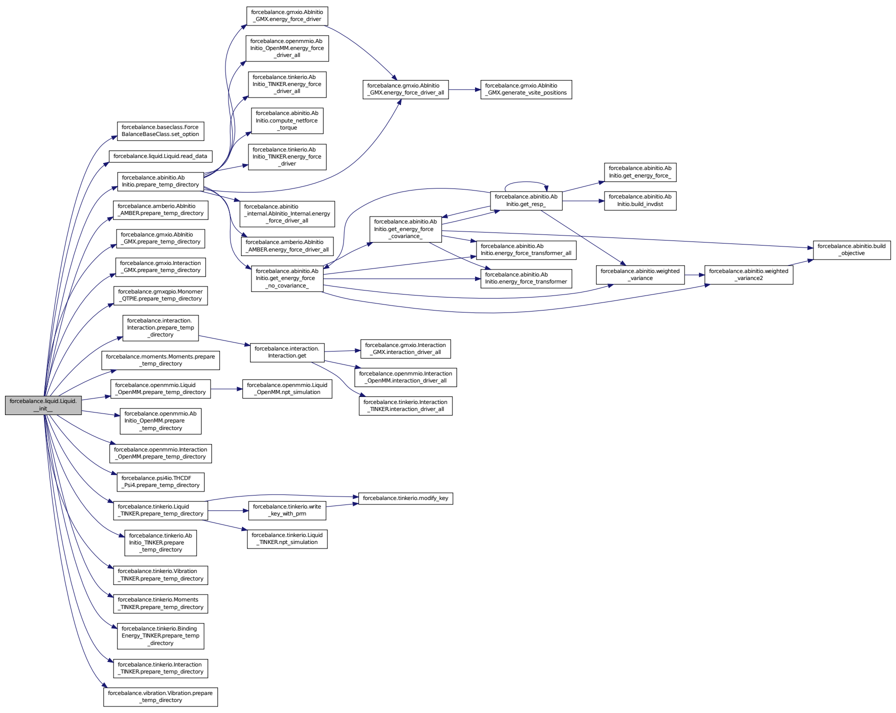
Instantiation of the subclass.

```
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.

Definition at line 58 of file liquid.py.

Here is the call graph for this function:



6.24.3 Member Function Documentation

6.24.3.1 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

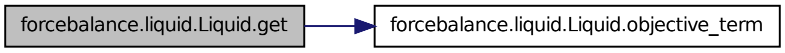
`in`	*mvals*	Mathematical parameter values
`in`	*AGrad*	Switch to turn on analytic gradient
`in`	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 336 of file liquid.py.

Here is the call graph for this function:


forcebalance.liquid.Liquid.get
forcebalance.liquid.Liquid.objective_term

6.24.4 Member Data Documentation

6.24.4.1 forcebalance.liquid.Liquid.MBarEnergy

Evaluated energies for all trajectories (i.e.

all iterations and all temperatures), using all mvals

Definition at line 106 of file liquid.py.

6.24.4.2 forcebalance.liquid.Liquid.SavedMVal

Read the reference data.

Prepare the temporary directory Saved force field mvals for all iterations

Definition at line 102 of file liquid.py.

6.24.4.3 forcebalance.liquid.Liquid.w_rho

Density.

Enthalpy of vaporization. Thermal expansion coefficient. Isothermal compressibility. Isobaric heat capacity. Static dielectric constant. Estimation of errors.

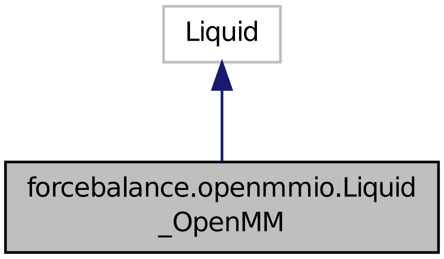
Definition at line 563 of file liquid.py.

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

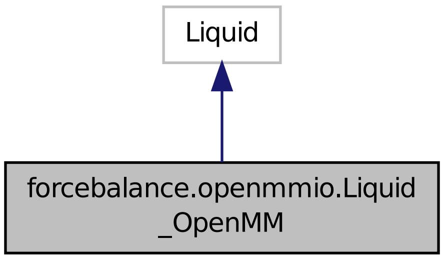
- liquid.py

6.25 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__**
- def prepare_temp_directory

  Prepare the temporary directory by copying in important files.
- def npt_simulation

  Submit a NPT simulation to the Work Queue.

6.25.1 Detailed Description

Definition at line 169 of file openmmio.py.

6.25.2 Member Function Documentation

6.25.2.1 def forcebalance.openmmio.Liquid_OpenMM.npt_simulation (*self*, *temperature*, *pressure*)

Submit a NPT simulation to the Work Queue.

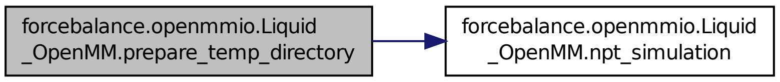
Definition at line 191 of file openmmio.py.

6.25.2.2 def forcebalance.openmmio.Liquid_OpenMM.prepare_temp_directory (*self*, *options*, *tgt_opts*)

Prepare the temporary directory by copying in important files.

Definition at line 181 of file openmmio.py.

Here is the call graph for this function:

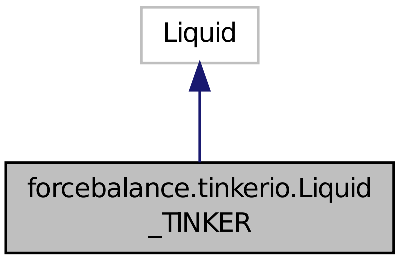

forcebalance.openmmio.Liquid
_OpenMM.prepare_temp_directory
forcebalance.openmmio.Liquid
_OpenMM.npt_simulation

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

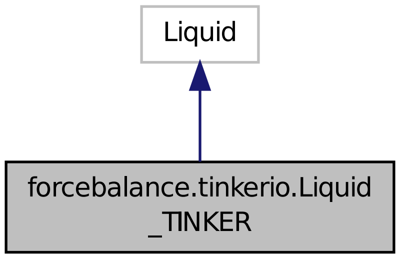
- openmmio.py

6.26 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__**
- 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**

6.26.1 Detailed Description

Definition at line 233 of file tinkerio.py.

6.26.2 Member Function Documentation

6.26.2.1 def forcebalance.tinkerio.Liquid_TINKER.npt_simulation (*self, temperature, pressure*)

Submit a NPT simulation to the Work Queue.

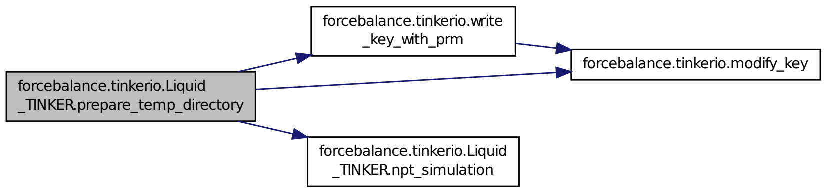
Definition at line 259 of file tinkerio.py.

6.26.2.2 def forcebalance.tinkerio.Liquid_TINKER.prepare_temp_directory (*self, options, tgt_opts*)

Prepare the temporary directory by copying in important files.

Definition at line 241 of file tinkerio.py.

Here is the call graph for this function:


forcebalance.tinkerio.Liquid
_TINKER.prepare_temp_directory
forcebalance.tinkerio.write
_key_with_prm
forcebalance.tinkerio.modify_key
forcebalance.tinkerio.Liquid
_TINKER.npt_simulation

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

- tinkerio.py

6.27 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**

6.27.1 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.

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

- Mol2.py

6.28 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**

6.28.1 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.

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

- Mol2.py

6.29 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**

6.29.1 Detailed Description

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

Definition at line 172 of file Mol2.py.

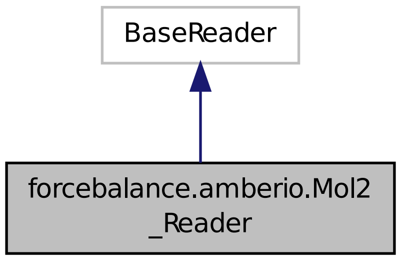
The documentation for this class was generated from the following file:

- Mol2.py

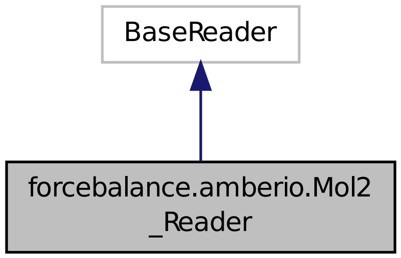
6.30 forcebalance.amberio.Mol2_Reader Class Reference

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**

6.30.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 40 of file amberio.py.

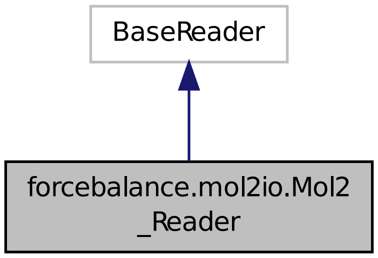
The documentation for this class was generated from the following file:

- amberio.py

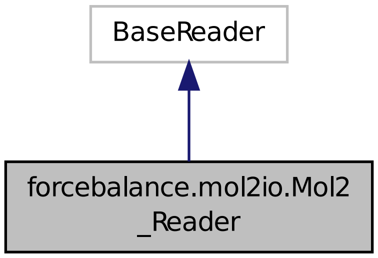
6.31 forcebalance.mol2io.Mol2_Reader Class Reference

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**

6.31.1 Detailed Description

Finite state machine for parsing Mol2 force field file.

(just for parameterizing the charges)

Definition at line 22 of file mol2io.py.

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

- mol2io.py

6.32 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**

6.32.1 Detailed Description

Definition at line 568 of file Mol2.py.

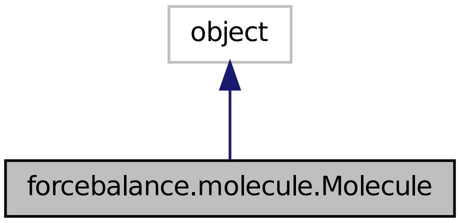
The documentation for this class was generated from the following file:

- Mol2.py

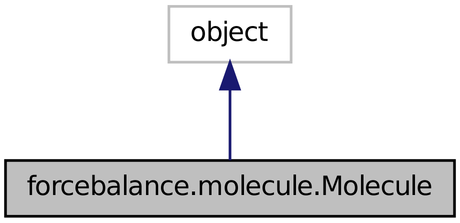
6.33 forcebalance.molecule.Molecule Class Reference

Lee-Ping's general file format conversion class.

Inheritance diagram for forcebalance.molecule.Molecule:



Collaboration diagram for forcebalance.molecule.Molecule:



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

  Reurn a copy of the object with certain atoms selected.
- 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.
- Data

  Creates entries like 'gromacs' : 'gromacs' and 'xyz' : 'xyz' in the Funnel.
- 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**

6.33.1 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 595 of file molecule.py.

6.33.2 Constructor & Destructor Documentation

6.33.2.1 def forcebalance.molecule.Molecule.__init__ (*self, fnm* = `None`, *ftype* = `None`)

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 762 of file molecule.py.

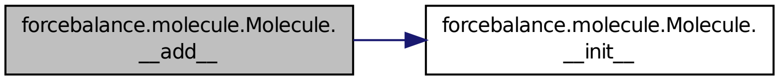
6.33.3 Member Function Documentation

6.33.3.1 def forcebalance.molecule.Molecule.__add__ (*self, other*)

Add method for Molecule objects.

Definition at line 701 of file molecule.py.

Here is the call graph for this function:

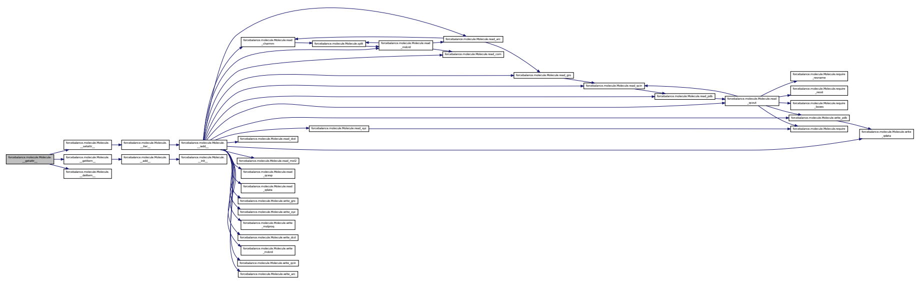

forcebalance.molecule.Molecule.
__add__
forcebalance.molecule.Molecule.
__init__

6.33.3.2 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 615 of file molecule.py.

Here is the call graph for this function:



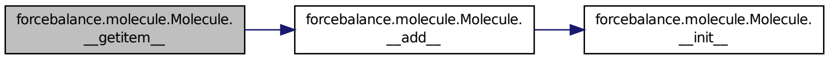
6.33.3.3 def forcebalance.molecule.Molecule.__getitem__ (*self*, *key*)

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 663 of file molecule.py.

Here is the call graph for this function:

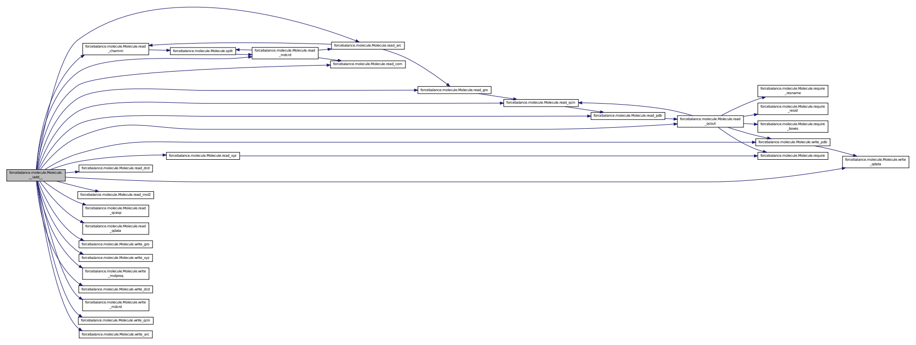

forcebalance.molecule.Molecule.
__getitem__
forcebalance.molecule.Molecule.
__add__
forcebalance.molecule.Molecule.
__init__

6.33.3.4 def forcebalance.molecule.Molecule.__iadd__ (*self*, *other*)

Add method for Molecule objects.

Definition at line 731 of file molecule.py.

Here is the call graph for this function:



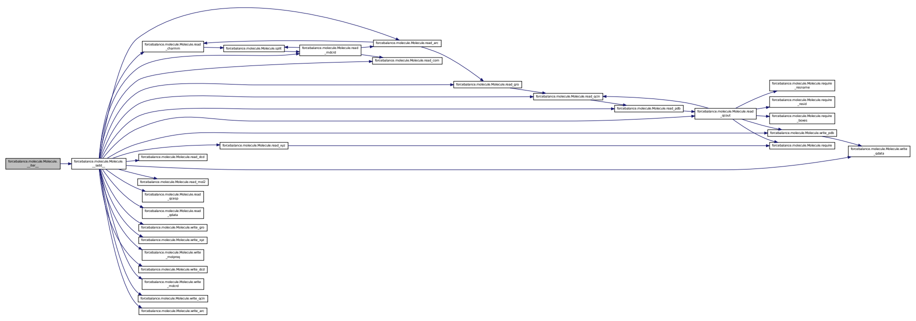
6.33.3.5 def forcebalance.molecule.Molecule.__iter__ (*self*)

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 690 of file molecule.py.

Here is the call graph for this function:



6.33.3.6 def forcebalance.molecule.Molecule.__len__ (*self*)

Return the number of frames in the trajectory.

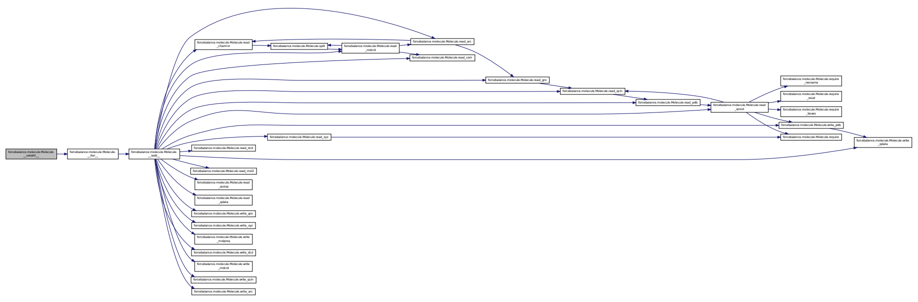
Definition at line 599 of file molecule.py.

6.33.3.7 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 651 of file molecule.py.

Here is the call graph for this function:



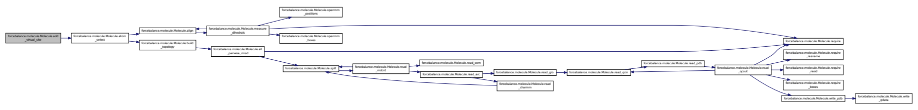
6.33.3.8 def forcebalance.molecule.Molecule.add_virtual_site (*self*, *idx*, *kwargs*)

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 954 of file molecule.py.

Here is the call graph for this function:



6.33.3.9 def forcebalance.molecule.Molecule.align (*self*, *smooth* = `True`, *center* = `True`)

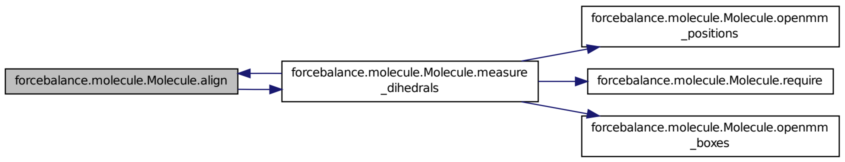
Align molecules.

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 1048 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.openmm_positions
forcebalance.molecule.Molecule.align
forcebalance.molecule.Molecule.measure_dihedrals
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.openmm_boxes

6.33.3.10 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 1029 of file molecule.py.

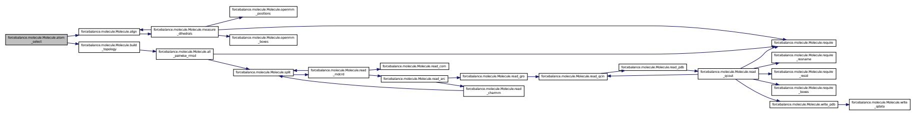
6.33.3.11 def forcebalance.molecule.Molecule.atom_select (*self*, *atomslice*)

Reurn a copy of the object with certain atoms selected.

Takes an integer, list or array as argument.

Definition at line 992 of file molecule.py.

Here is the call graph for this function:



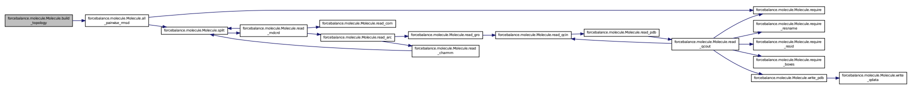
6.33.3.12 def forcebalance.molecule.Molecule.build_topology (*self*, *sn* = `None`, *Fac* = `1.2`)

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 1067 of file molecule.py.

Here is the call graph for this function:

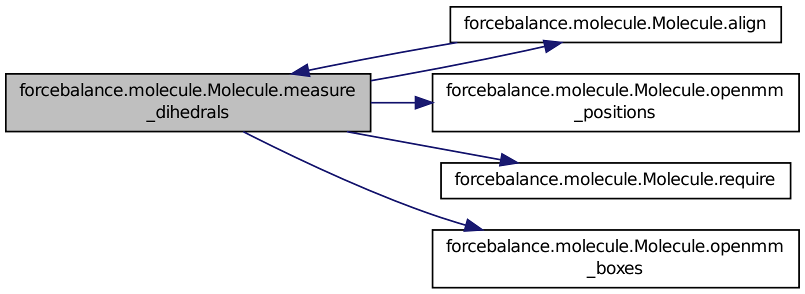


6.33.3.13 def forcebalance.molecule.Molecule.measure_dihedrals (*self*, *i*, *j*, *k*, *l*)

Return a series of dihedral angles, given four atom indices numbered from zero.

Definition at line 1102 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.align
forcebalance.molecule.Molecule.measure_dihedrals
forcebalance.molecule.Molecule.openmm_positions
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.openmm_boxes

6.33.3.14 def forcebalance.molecule.Molecule.read_arc (*self, fnm*)

Read a TINKER .arc file.

Parameters

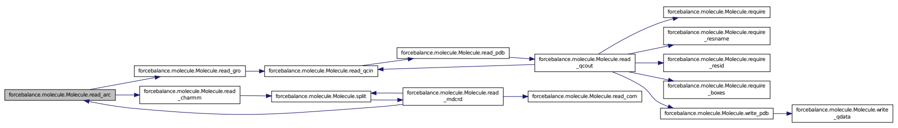
in	*fnm*	The input file name

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 1446 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.read_arc
forcebalance.molecule.Molecule.read_gro
forcebalance.molecule.Molecule.read_charmm
forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.split
forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_mdcrd
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.read_com
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.15 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 1403 of file molecule.py.

6.33.3.16 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

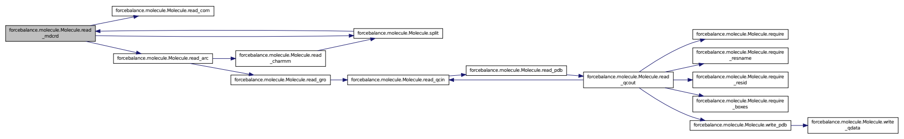
in	*fnm*	The input file name

Returns

xyzs A list of XYZ coordinates (number of snapshots times number of atoms)
boxes Boxes (if present.)

Definition at line 1257 of file molecule.py.

Here is the call graph for this function:

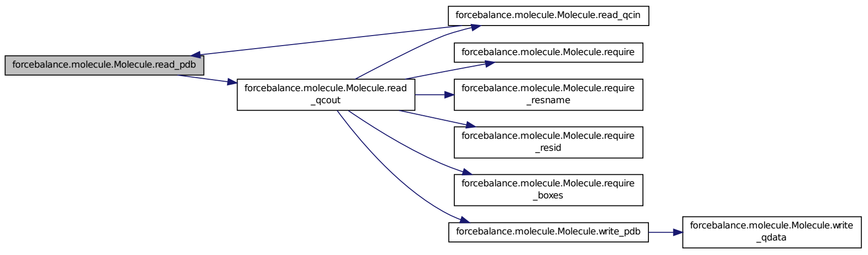

forcebalance.molecule.Molecule.read_mdcrd
forcebalance.molecule.Molecule.read_com
forcebalance.molecule.Molecule.split
forcebalance.molecule.Molecule.read_arc
forcebalance.molecule.Molecule.read_charmm
forcebalance.molecule.Molecule.read_gro
forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.17 def forcebalance.molecule.Molecule.read_pdb (*self, fnm*)

Loads a PDB and returns a dictionary containing its data.

Definition at line 1735 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.18 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.

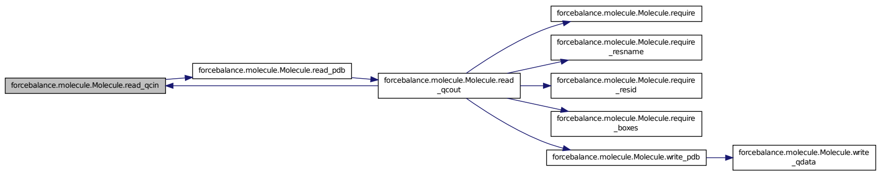
2) When we print the Q-Chem file, we should preserve the line ordering of the 'rem' section, but also be able to add 'rem' options at the end.

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 1628 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.19 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:

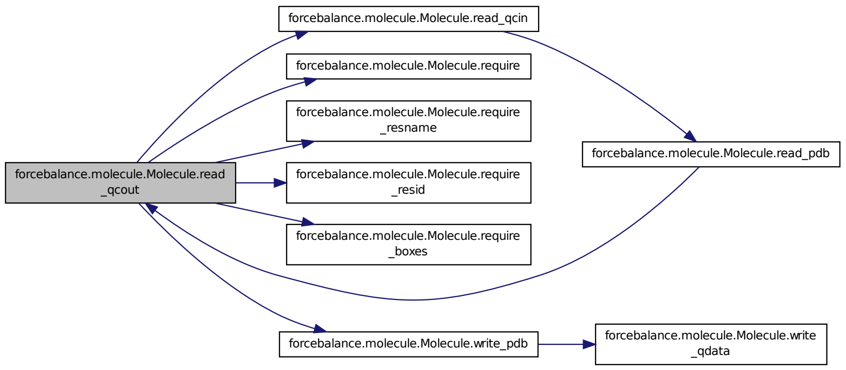
- Coordinates
- Energies
- Forces

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 1814 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.20 def forcebalance.molecule.Molecule.read_xyz (*self*, *fnm*)

Parse a .xyz file which contains several xyz coordinates, and return their elements.

Parameters

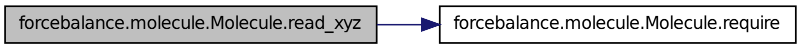
in	*fnm*	The input file name

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 1212 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.read_xyz
forcebalance.molecule.Molecule.require

6.33.3.21 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 971 of file molecule.py.

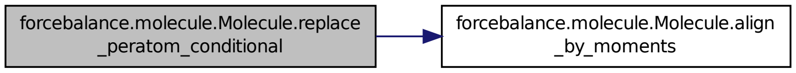
6.33.3.22 def forcebalance.molecule.Molecule.replace_peratom_conditional (*self, key1, cond, key2, orig, want*)

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 982 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.replace_peratom_conditional
forcebalance.molecule.Molecule.align_by_moments

6.33.3.23 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.

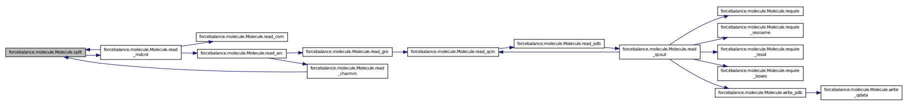
If the number of chunks / frames is not specified, then one file is written for each frame.

Returns

fnms A list of the file names that were written.

Definition at line 1194 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.split
forcebalance.molecule.Molecule.read_mdcrd
forcebalance.molecule.Molecule.read_com
forcebalance.molecule.Molecule.read_arc
forcebalance.molecule.Molecule.read_charmm
forcebalance.molecule.Molecule.read_gro
forcebalance.molecule.Molecule.read_qcin
forcebalance.molecule.Molecule.read_pdb
forcebalance.molecule.Molecule.read_qcout
forcebalance.molecule.Molecule.require
forcebalance.molecule.Molecule.require_resname
forcebalance.molecule.Molecule.require_resid
forcebalance.molecule.Molecule.require_boxes
forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.24 def forcebalance.molecule.Molecule.write_pdb (*self*, *select*)

Save to a PDB.

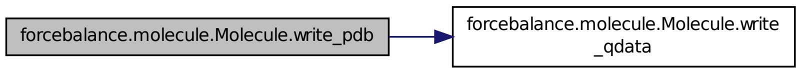
Copied wholesale from MSMBuilder.

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.

Definition at line 2146 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.Molecule.write_pdb
forcebalance.molecule.Molecule.write_qdata

6.33.3.25 def forcebalance.molecule.Molecule.write_qdata (*self*, *select*)

Text quantum data format.

Definition at line 2214 of file molecule.py.

6.33.4 Member Data Documentation

6.33.4.1 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.

6.33.4.2 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 876 of file molecule.py.

6.33.4.3 forcebalance.molecule.Molecule.Funnel

A funnel dictionary that takes redundant file types and maps them down to a few.

Definition at line 794 of file molecule.py.

6.33.4.4 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 837 of file molecule.py.

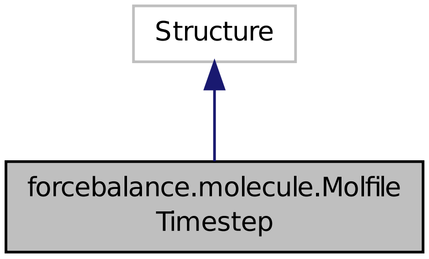
The documentation for this class was generated from the following file:

- molecule.py

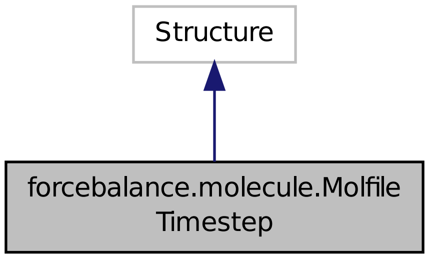
6.34 forcebalance.molecule.MolfileTimestep Class Reference

Wrapper for the timestep C structure used in molfile plugins.

Inheritance diagram for forcebalance.molecule.MolfileTimestep:



Collaboration diagram for forcebalance.molecule.MolfileTimestep:



6.34.1 Detailed Description

Wrapper for the timestep C structure used in molfile plugins.

Definition at line 424 of file molecule.py.

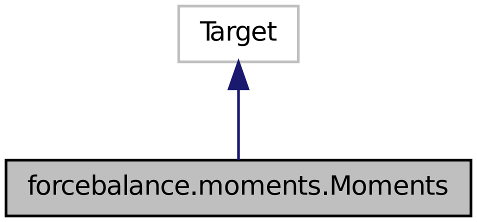
The documentation for this class was generated from the following file:

- molecule.py

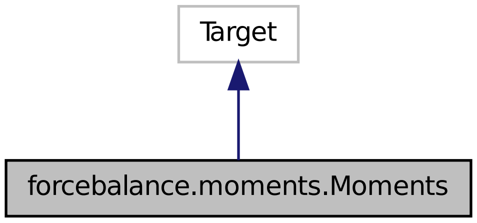
6.35 forcebalance.moments.Moments Class Reference

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**

6.35.1 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.

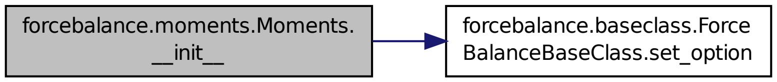
6.35.2 Constructor & Destructor Documentation

6.35.2.1 def forcebalance.moments.Moments.__init__ (*self*, *options*, *tgt_opts*, *forcefield*)

Initialization.

Definition at line 32 of file moments.py.

Here is the call graph for this function:


forcebalance.moments.Moments.__init__
forcebalance.baseclass.ForceBalanceBaseClass.set_option

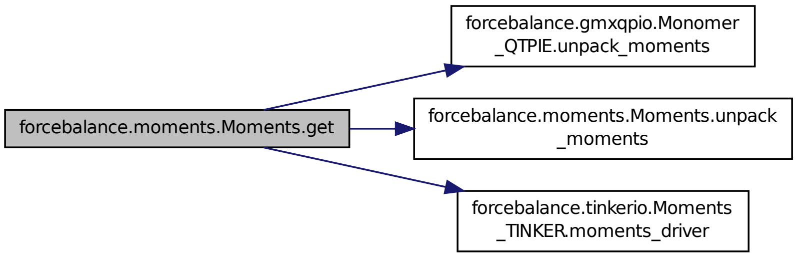
6.35.3 Member Function Documentation

6.35.3.1 def forcebalance.moments.Moments.get (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

Definition at line 170 of file moments.py.

Here is the call graph for this function:

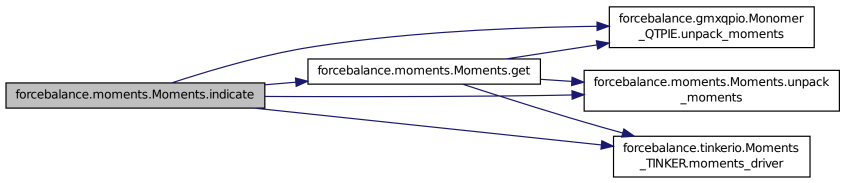

forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.moments.Moments.get
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver

6.35.3.2 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:


forcebalance.moments.Moments.indicate
forcebalance.moments.Moments.get
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver

6.35.3.3 def forcebalance.moments.Moments.prepare_temp_directory (*self*, *options*, *tgt_opts*)

Prepare the temporary directory.

Definition at line 133 of file moments.py.

6.35.3.4 def forcebalance.moments.Moments.read_reference_data (*self*)

Read the reference data from a file.

Definition at line 64 of file moments.py.

6.35.4 Member Data Documentation

6.35.4.1 forcebalance.moments.Moments.mfnm

The mdata.txt file that contains the moments.

Definition at line 54 of file moments.py.

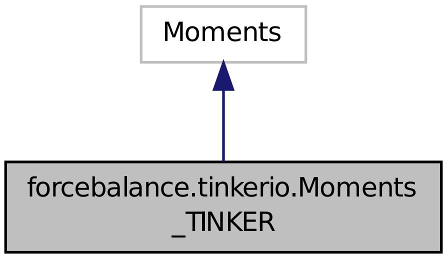
The documentation for this class was generated from the following file:

- moments.py

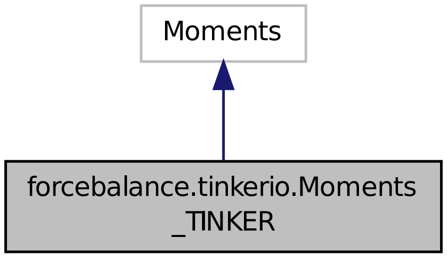
6.36 forcebalance.tinkerio.Moments_TINKER Class Reference

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__**
- def **prepare_temp_directory**
- def **moments_driver**

6.36.1 Detailed Description

Subclass of Target for multipole moment matching using TINKER.

Definition at line 398 of file tinkerio.py.

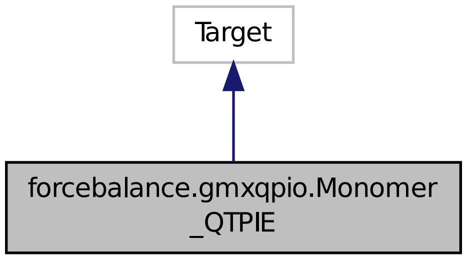
The documentation for this class was generated from the following file:

- tinkerio.py

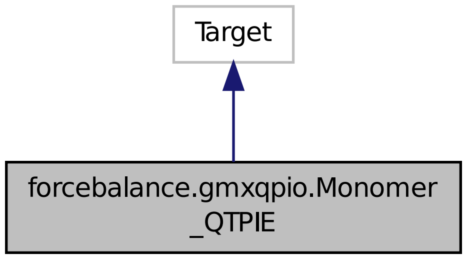
6.37 forcebalance.gmxqpio.Monomer_QTPIE Class Reference

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__**
- def indicate

    Print qualitative indicator.
- def **prepare_temp_directory**
- def **unpack_moments**
- def **get**

Public Attributes

- **ref_moments**

- **weights**
- **calc_moments**
- **objective**

6.37.1 Detailed Description

Subclass of Target for monomer properties of QTPIE (implemented within gromacs WCV branch).

Definition at line 127 of file gmxqpio.py.

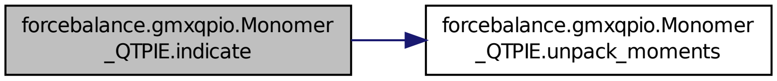
6.37.2 Member Function Documentation

6.37.2.1 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:

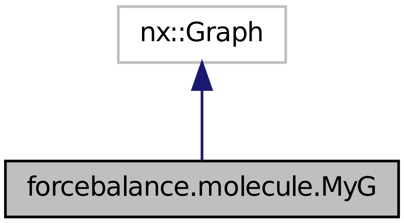

forcebalance.gmxqpio.Monomer_QTPIE.indicate
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments

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

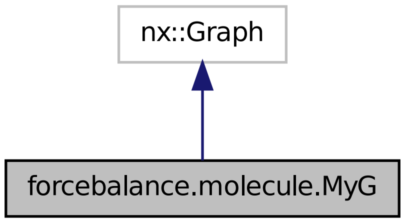
- gmxqpio.py

6.38 forcebalance.molecule.MyG Class Reference

Inheritance diagram for forcebalance.molecule.MyG:



Collaboration diagram for forcebalance.molecule.MyG:



Public Member Functions

- def **__init__**
- def **__eq__**
- def __hash__

  The hash function is something we can use to discard two things that are obviously not equal.
- def L

  Return a list of the sorted atom numbers in this graph.
- def AStr

  Return a string of atoms, which serves as a rudimentary 'fingerprint' : '99,100,103,151' .
- def e

  Return an array of the elements.
- def ef

  Create an Empirical Formula.
- def x

  Get a list of the coordinates.

Public Attributes

- **Alive**

6.38.1 Detailed Description

Definition at line 255 of file molecule.py.

6.38.2 Member Function Documentation

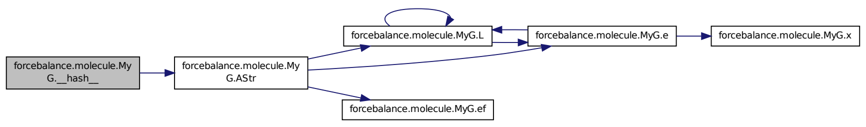
6.38.2.1 def forcebalance.molecule.MyG.__hash__ (*self*)

The hash function is something we can use to discard two things that are obviously not equal.

Here we neglect the hash.

Definition at line 268 of file molecule.py.

Here is the call graph for this function:

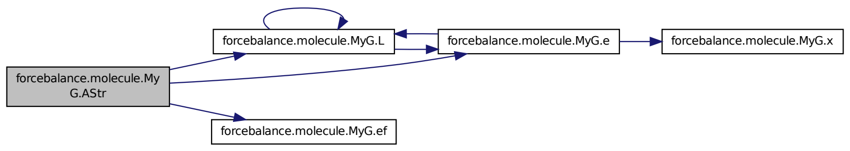

forcebalance.molecule.MyG.__hash__
forcebalance.molecule.MyG.AStr
forcebalance.molecule.MyG.L
forcebalance.molecule.MyG.e
forcebalance.molecule.MyG.x
forcebalance.molecule.MyG.ef

6.38.2.2 def forcebalance.molecule.MyG.AStr (*self*)

Return a string of atoms, which serves as a rudimentary 'fingerprint' : '99,100,103,151' .

Definition at line 276 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.MyG.AStr
forcebalance.molecule.MyG.L
forcebalance.molecule.MyG.e
forcebalance.molecule.MyG.x
forcebalance.molecule.MyG.ef

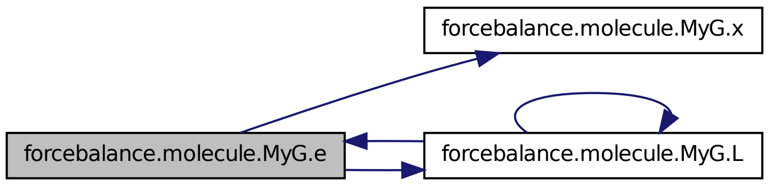
6.38.2.3 def forcebalance.molecule.MyG.e (*self*)

Return an array of the elements.

For instance ['H' 'C' 'C' 'H'].

Definition at line 280 of file molecule.py.

Here is the call graph for this function:

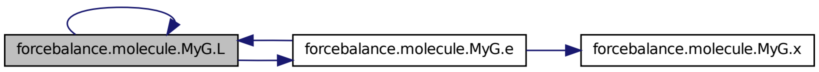

forcebalance.molecule.MyG.x
forcebalance.molecule.MyG.e
forcebalance.molecule.MyG.L

6.38.2.4 def forcebalance.molecule.MyG.L (*self*)

Return a list of the sorted atom numbers in this graph.

Definition at line 272 of file molecule.py.

Here is the call graph for this function:


forcebalance.molecule.MyG.L
forcebalance.molecule.MyG.e
forcebalance.molecule.MyG.x

6.38.2.5 def forcebalance.molecule.MyG.x (*self*)

Get a list of the coordinates.

Definition at line 290 of file molecule.py.

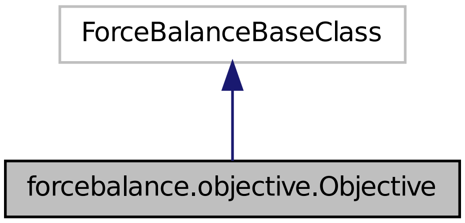
The documentation for this class was generated from the following file:

- molecule.py

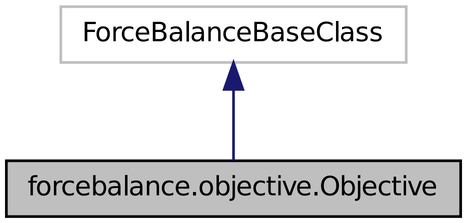
6.39 forcebalance.objective.Objective Class Reference

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**

6.39.1 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 36 of file objective.py.

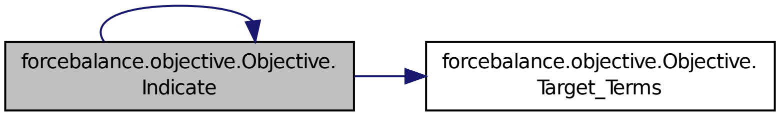
6.39.2 Member Function Documentation

6.39.2.1 def forcebalance.objective.Objective.Indicate (*self*)

Print objective function contributions.

Definition at line 141 of file objective.py.

Here is the call graph for this function:


forcebalance.objective.Objective.
Indicate
forcebalance.objective.Objective.
Target_Terms

6.39.3 Member Data Documentation

6.39.3.1 forcebalance.objective.Objective.Penalty

Initialize the penalty function.

Definition at line 62 of file objective.py.

6.39.3.2 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 52 of file objective.py.

6.39.3.3 forcebalance.objective.Objective.WTot

Obtain the denominator.

Definition at line 67 of file objective.py.

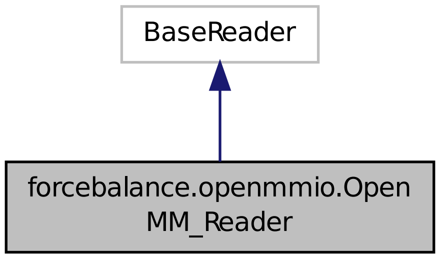
The documentation for this class was generated from the following file:

- objective.py

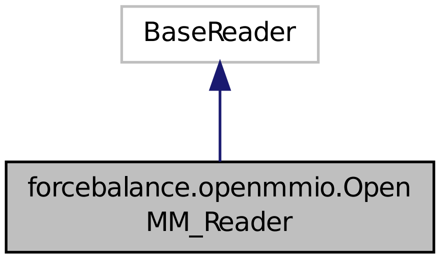
6.40 forcebalance.openmmio.OpenMM_Reader Class Reference

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.

6.40.1 Detailed Description

Class for parsing OpenMM force field files.

Definition at line 144 of file openmmio.py.

6.40.2 Member Function Documentation

6.40.2.1 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 154 of file openmmio.py.

6.40.3 Member Data Documentation

6.40.3.1 forcebalance.openmmio.OpenMM_Reader.pdict

Initialize the superclass.

:) The parameter dictionary (defined in this file)

Definition at line 149 of file openmmio.py.

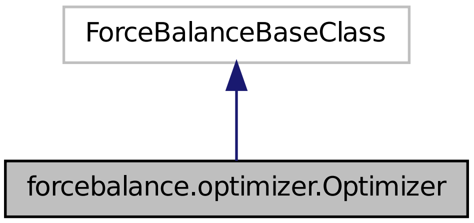
The documentation for this class was generated from the following file:

- openmmio.py

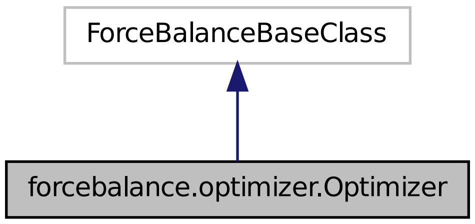
6.41 forcebalance.optimizer.Optimizer Class Reference

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**

6.41.1 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.

6.41.2 Constructor & Destructor Documentation

6.41.2.1 def forcebalance.optimizer.Optimizer.__init__ (*self,* *options,* *Objective,* *FF*)

Create an Optimizer object.

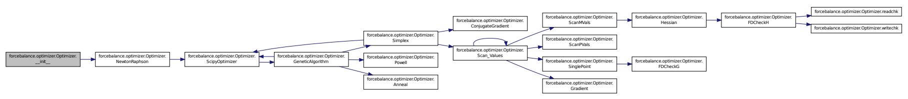
The optimizer depends on both the FF and the fitting targets so there is a chain of dependencies: FF –> FitSim –> Optimizer, and FF –> Optimizer

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.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.__init__
forcebalance.optimizer.Optimizer.NewtonRaphson
forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.Gradient
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3 Member Function Documentation

6.41.3.1 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 778 of file optimizer.py.

6.41.3.2 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 598 of file optimizer.py.

6.41.3.3 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 783 of file optimizer.py.

6.41.3.4 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 879 of file optimizer.py.

6.41.3.5 def forcebalance.optimizer.Optimizer.FDCheckH (*self*)

Finite-difference checker for the objective function Hessian.

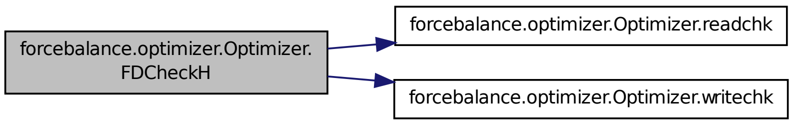
For each element in the Hessian, use a five-point stencil in both parameter indices to compute a finite-difference derivative, and compare it to the analytic result.

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 911 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.
FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.6 def forcebalance.optimizer.Optimizer.GeneticAlgorithm (*self*)

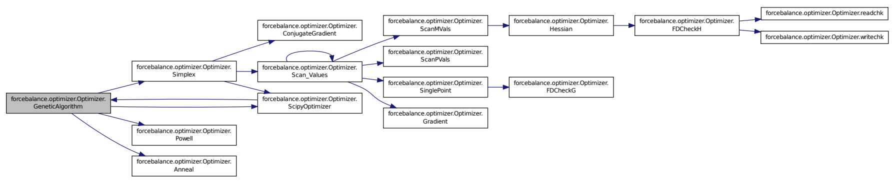
Genetic algorithm, under development.

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 675 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.Gradient
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.7 def forcebalance.optimizer.Optimizer.Gradient (*self*)

A single-point gradient computation.

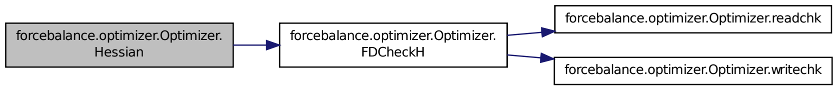
Definition at line 857 of file optimizer.py.

6.41.3.8 def forcebalance.optimizer.Optimizer.Hessian (*self*)

A single-point Hessian computation.

Definition at line 865 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.
Hessian
forcebalance.optimizer.Optimizer.
FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.9 def forcebalance.optimizer.Optimizer.MainOptimizer (*self,* *b_BFGS* = 0)

The main ForceBalance adaptive trust-radius pseudo-Newton optimizer.

Tried and true in many situations. :)

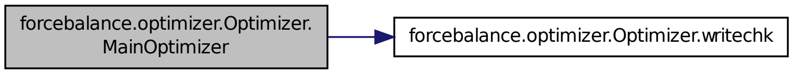
```
Usually this function is called with the BFGS or NewtonRaphson
method.  The NewtonRaphson method is consistently the best
method I have, because I always provide at least an
approximate Hessian to the objective function.  The BFGS
method is vestigial and currently does not work.

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 226 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.
MainOptimizer
forcebalance.optimizer.Optimizer.writechk

6.41.3.10 def forcebalance.optimizer.Optimizer.NewtonRaphson (*self*)

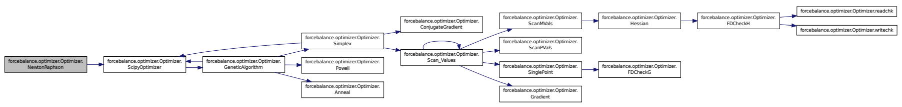
Optimize the force field parameters using the Newton-Raphson method (.

See Also

MainOptimizer)

Definition at line 593 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.NewtonRaphson
forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.Gradient
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.11 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 773 of file optimizer.py.

6.41.3.12 def forcebalance.optimizer.Optimizer.readchk (*self*)

Read the checkpoint file for the main optimizer.

Definition at line 939 of file optimizer.py.

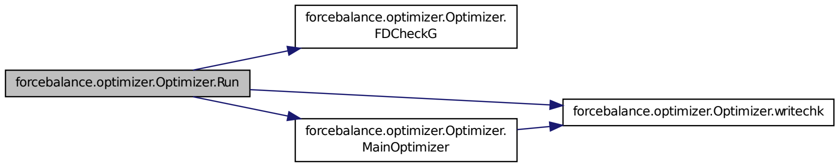
6.41.3.13 def forcebalance.optimizer.Optimizer.Run (*self*)

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:


forcebalance.optimizer.Optimizer.
FDCheckG
forcebalance.optimizer.Optimizer.Run
forcebalance.optimizer.Optimizer.writechk
forcebalance.optimizer.Optimizer.
MainOptimizer

6.41.3.14 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.

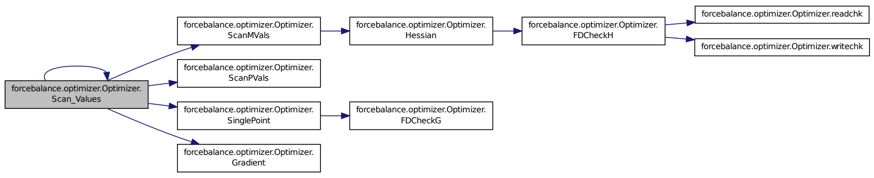
Todo Maybe a multidimensional grid can be done.

Parameters

`in`	*MathPhys*	Switch to use mathematical (True) or physical (False) parameters.

Definition at line 809 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.
Scan_Values
forcebalance.optimizer.Optimizer.
ScanMVals
forcebalance.optimizer.Optimizer.
ScanPVals
forcebalance.optimizer.Optimizer.
SinglePoint
forcebalance.optimizer.Optimizer.
Gradient
forcebalance.optimizer.Optimizer.
Hessian
forcebalance.optimizer.Optimizer.
FDCheckG
forcebalance.optimizer.Optimizer.
FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.15 def forcebalance.optimizer.Optimizer.ScanMVals (*self*)

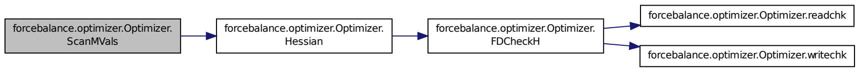
Scan through the mathematical parameter space.

See Also

Optimizer::ScanValues

Definition at line 841 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.16 def forcebalance.optimizer.Optimizer.ScanPVals (*self*)

Scan through the physical parameter space.

See Also

Optimizer::ScanValues

Definition at line 846 of file optimizer.py.

6.41.3.17 def forcebalance.optimizer.Optimizer.ScipyOptimizer (*self, Algorithm* = `"None"`)

Driver for SciPy optimizations.

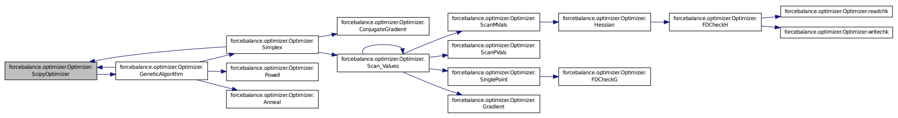
```
    Using any of the SciPy optimizers requires that SciPy is installed.
    This method first defines several wrappers around the objective function that the SciPy
    optimizers can use.  Then it calls the algorith mitself.
```

Parameters

in	*Algorithm*	The optimization algorithm to use, for example 'powell', 'simplex' or 'anneal'

Definition at line 611 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.Gradient
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk

6.41.3.18 def forcebalance.optimizer.Optimizer.Simplex (*self*)

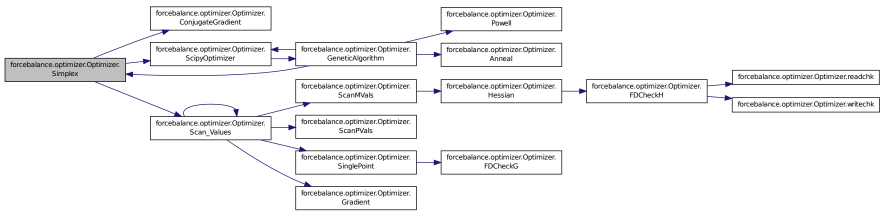
Use SciPy's built-in simplex algorithm to optimize the parameters.

See Also

Optimizer::ScipyOptimizer

Definition at line 768 of file optimizer.py.

Here is the call graph for this function:

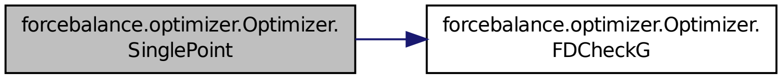

forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.Gradient

6.41.3.19 def forcebalance.optimizer.Optimizer.SinglePoint (*self*)

A single-point objective function computation.

Definition at line 851 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.FDCheckG

6.41.3.20 def forcebalance.optimizer.Optimizer.step (*self, xk, data, trust*)

Computes the next step in the parameter space.

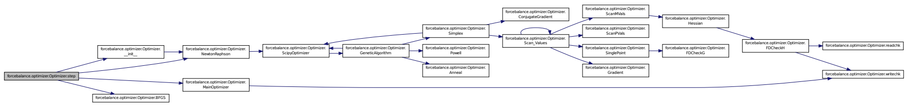
There are lots of tricks here that I will document later.

Parameters

`in`	*G*	The gradient
`in`	*H*	The Hessian
`in`	*trust*	The trust radius

Definition at line 418 of file optimizer.py.

Here is the call graph for this function:


forcebalance.optimizer.Optimizer.step
forcebalance.optimizer.Optimizer.__init__
forcebalance.optimizer.Optimizer.NewtonRaphson
forcebalance.optimizer.Optimizer.ScipyOptimizer
forcebalance.optimizer.Optimizer.GeneticAlgorithm
forcebalance.optimizer.Optimizer.Simplex
forcebalance.optimizer.Optimizer.ConjugateGradient
forcebalance.optimizer.Optimizer.Scan_Values
forcebalance.optimizer.Optimizer.ScanMVals
forcebalance.optimizer.Optimizer.Hessian
forcebalance.optimizer.Optimizer.ScanPVals
forcebalance.optimizer.Optimizer.SinglePoint
forcebalance.optimizer.Optimizer.FDCheckG
forcebalance.optimizer.Optimizer.FDCheckH
forcebalance.optimizer.Optimizer.readchk
forcebalance.optimizer.Optimizer.writechk
forcebalance.optimizer.Optimizer.Powell
forcebalance.optimizer.Optimizer.Anneal
forcebalance.optimizer.Optimizer.Gradient
forcebalance.optimizer.Optimizer.MainOptimizer
forcebalance.optimizer.Optimizer.BFGS

6.41.3.21 def forcebalance.optimizer.Optimizer.writechk (*self*)

Write the checkpoint file for the main optimizer.

Definition at line 951 of file optimizer.py.

6.41.4 Member Data Documentation

6.41.4.1 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 153 of file optimizer.py.

6.41.4.2 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 137 of file optimizer.py.

6.41.4.3 forcebalance.optimizer.Optimizer.OptTab

A list of all the things we can ask the optimizer to do.

Definition at line 59 of file optimizer.py.

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

- optimizer.py

6.42 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**
- **Pen_Names**
- **ptyp**
- **Pen_Tab**
- spacings

  Find exponential spacings.

6.42.1 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 235 of file objective.py.

6.42.2 Member Function Documentation

6.42.2.1 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 318 of file objective.py.

6.42.2.2 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

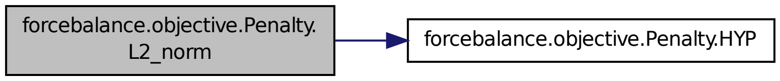
in	*mvals*	The parameter vector

Returns

DC0 The norm squared of the vector
DC1 The gradient of DC0
DC2 The Hessian (just a constant)

Definition at line 298 of file objective.py.

Here is the call graph for this function:


forcebalance.objective.Penalty.
L2_norm
forcebalance.objective.Penalty.HYP

6.42.3 Member Data Documentation

6.42.3.1 forcebalance.objective.Penalty.spacings

Find exponential spacings.

Definition at line 263 of file objective.py.

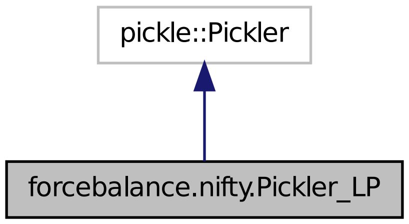
The documentation for this class was generated from the following file:

- objective.py

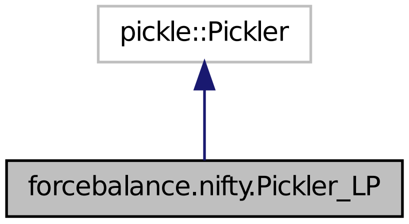
6.43 forcebalance.nifty.Pickler_LP Class Reference

A subclass of the python Pickler that implements pickling of _ElementTree types.

Inheritance diagram for forcebalance.nifty.Pickler_LP:



Collaboration diagram for forcebalance.nifty.Pickler_LP:



Public Member Functions

- def **__init__**

6.43.1 Detailed Description

A subclass of the python Pickler that implements pickling of _ElementTree types.

Definition at line 383 of file nifty.py.

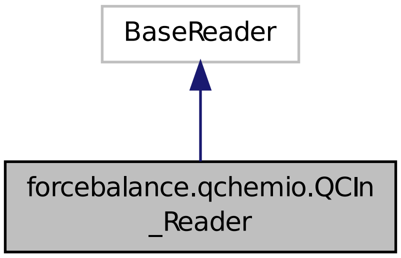
The documentation for this class was generated from the following file:

- nifty.py

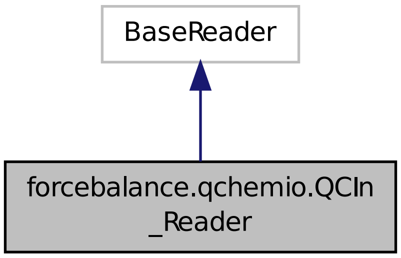
6.44 forcebalance.qchemio.QCIn_Reader Class Reference

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**

6.44.1 Detailed Description

Finite state machine for parsing Q-Chem input files.

Definition at line 28 of file qchemio.py.

6.44.2 Member Function Documentation

6.44.2.1 def forcebalance.qchemio.QCIn_Reader.feed (*self, line*)

Feed in a line.

Parameters

`in`	*line*	The line of data

Definition at line 45 of file qchemio.py.

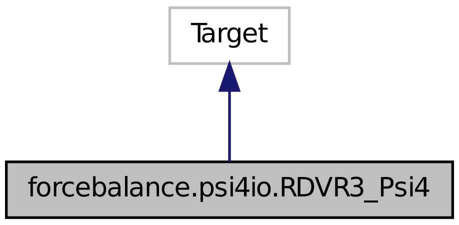
The documentation for this class was generated from the following file:

- qchemio.py

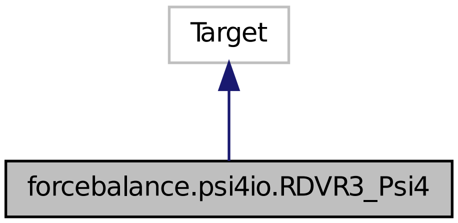
6.45 forcebalance.psi4io.RDVR3_Psi4 Class Reference

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__**
- 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**

6.45.1 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.

6.45.2 Member Function Documentation

6.45.2.1 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

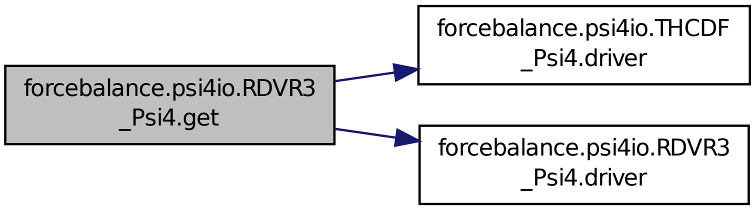
`in`	*mvals*	Mathematical parameter values
`in`	*AGrad*	Switch to turn on analytic gradient
`in`	*AHess*	Switch to turn on analytic Hessian

Returns

Answer Contribution to the objective function

Definition at line 382 of file psi4io.py.

Here is the call graph for this function:


forcebalance.psi4io.RDVR3_Psi4.get
forcebalance.psi4io.THCDF_Psi4.driver
forcebalance.psi4io.RDVR3_Psi4.driver

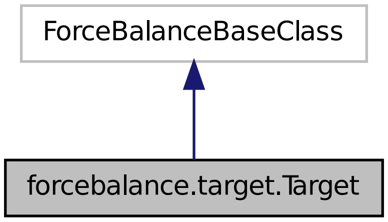
The documentation for this class was generated from the following file:

- psi4io.py

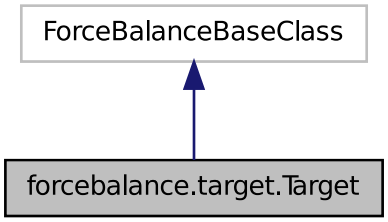
6.46 forcebalance.target.Target Class Reference

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.

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.

6.46.1 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.

6.46.2 Constructor & Destructor Documentation

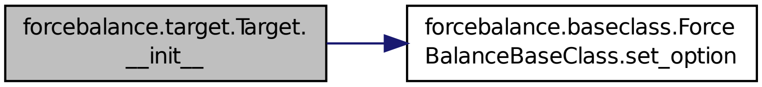
6.46.2.1 def forcebalance.target.Target.__init__ (*self*, *options*, *tgt_opts*, *forcefield*)

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.

Definition at line 89 of file target.py.

Here is the call graph for this function:


forcebalance.target.Target.
__init__
forcebalance.baseclass.Force
BalanceBaseClass.set_option

6.46.3 Member Function Documentation

6.46.3.1 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.

Definition at line 251 of file target.py.

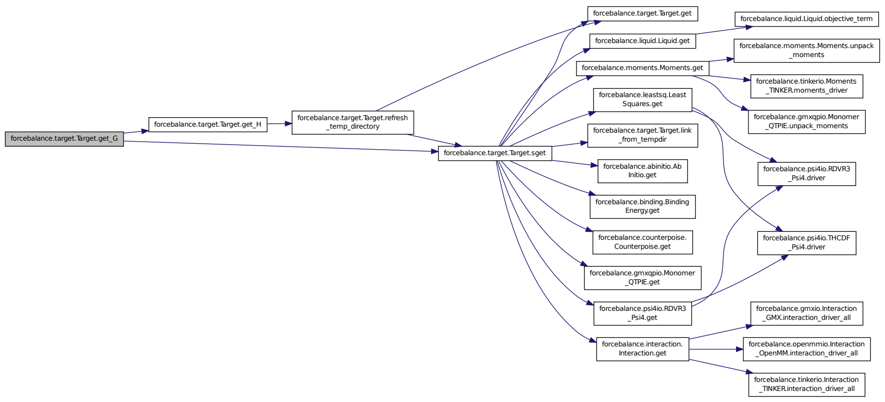
6.46.3.2 def forcebalance.target.Target.get_G (*self*, *mvals* = `None`)

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:


forcebalance.target.Target.get_G
forcebalance.target.Target.get_H
forcebalance.target.Target.refresh_temp_directory
forcebalance.target.Target.sget
forcebalance.target.Target.get
forcebalance.liquid.Liquid.get
forcebalance.moments.Moments.get
forcebalance.leastsq.LeastSquares.get
forcebalance.target.Target.link_from_tempdir
forcebalance.abinitio.AbInitio.get
forcebalance.binding.BindingEnergy.get
forcebalance.counterpoise.Counterpoise.get
forcebalance.gmxqpio.Monomer_QTPIE.get
forcebalance.psi4io.RDVR3_Psi4.get
forcebalance.interaction.Interaction.get
forcebalance.liquid.Liquid.objective_term
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.psi4io.RDVR3_Psi4.driver
forcebalance.psi4io.THCDF_Psi4.driver
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all

6.46.3.3 def forcebalance.target.Target.get_H (*self*, *mvals* = `None`)

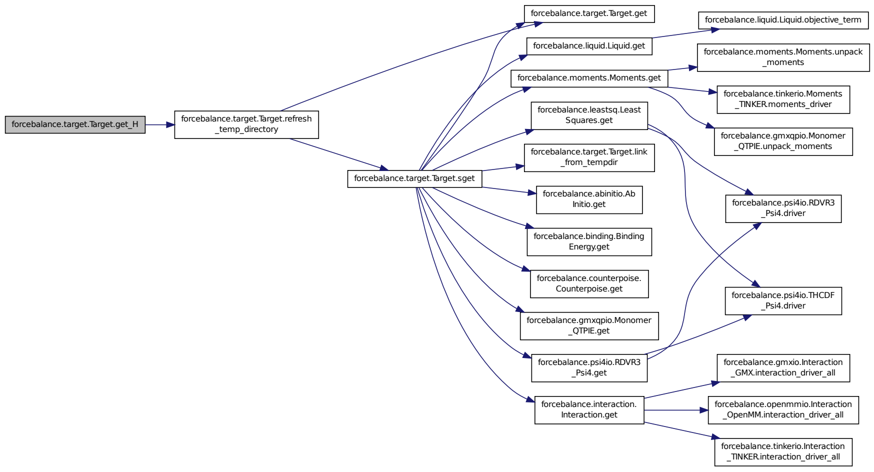
Computes the objective function contribution and its gradient / Hessian.

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:

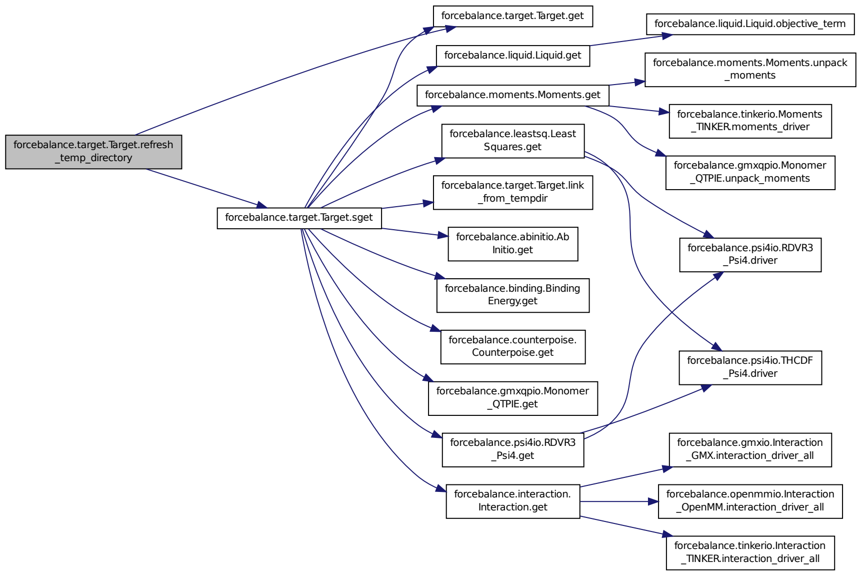

forcebalance.target.Target.get_H
forcebalance.target.Target.refresh_temp_directory
forcebalance.target.Target.sget
forcebalance.target.Target.get
forcebalance.liquid.Liquid.get
forcebalance.moments.Moments.get
forcebalance.leastsq.LeastSquares.get
forcebalance.target.Target.link_from_tempdir
forcebalance.abinitio.AbInitio.get
forcebalance.binding.BindingEnergy.get
forcebalance.counterpoise.Counterpoise.get
forcebalance.gmxqpio.Monomer_QTPIE.get
forcebalance.psi4io.RDVR3_Psi4.get
forcebalance.interaction.Interaction.get
forcebalance.liquid.Liquid.objective_term
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.psi4io.RDVR3_Psi4.driver
forcebalance.psi4io.THCDF_Psi4.driver
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all

6.46.3.4 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:


forcebalance.target.Target.refresh_temp_directory
forcebalance.target.Target.sget
forcebalance.target.Target.get
forcebalance.liquid.Liquid.get
forcebalance.moments.Moments.get
forcebalance.leastsq.LeastSquares.get
forcebalance.target.Target.link_from_tempdir
forcebalance.abinitio.AbInitio.get
forcebalance.binding.BindingEnergy.get
forcebalance.counterpoise.Counterpoise.get
forcebalance.gmxqpio.Monomer_QTPIE.get
forcebalance.psi4io.RDVR3_Psi4.get
forcebalance.interaction.Interaction.get
forcebalance.liquid.Liquid.objective_term
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.psi4io.RDVR3_Psi4.driver
forcebalance.psi4io.THCDF_Psi4.driver
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all

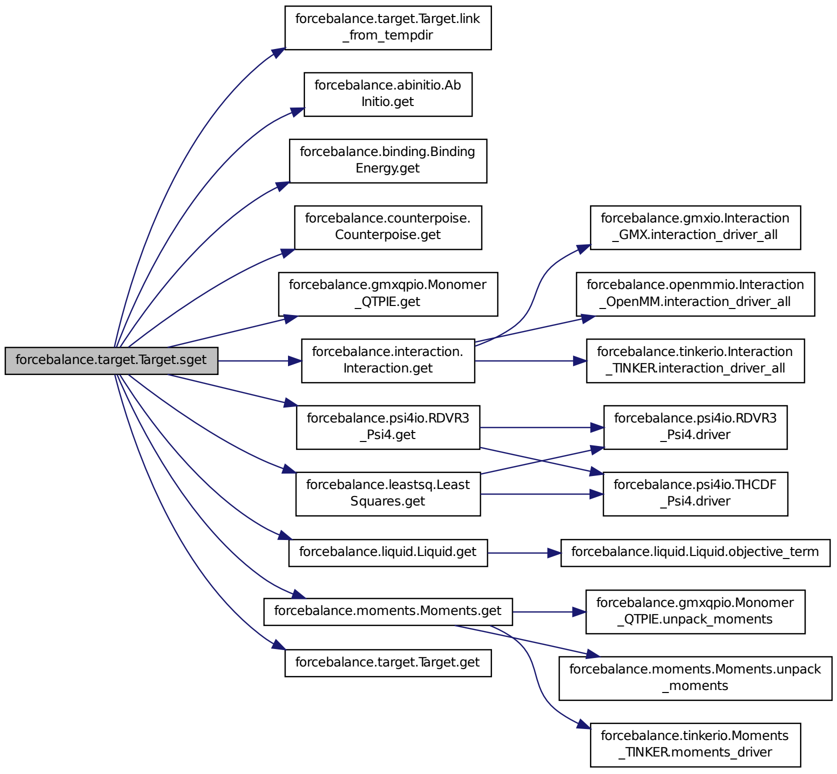
6.46.3.5 def forcebalance.target.Target.sget (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`)

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:


forcebalance.target.Target.sget
forcebalance.target.Target.link_from_tempdir
forcebalance.abinitio.AbInitio.get
forcebalance.binding.BindingEnergy.get
forcebalance.counterpoise.Counterpoise.get
forcebalance.gmxqpio.Monomer_QTPIE.get
forcebalance.interaction.Interaction.get
forcebalance.psi4io.RDVR3_Psi4.get
forcebalance.leastsq.LeastSquares.get
forcebalance.liquid.Liquid.get
forcebalance.moments.Moments.get
forcebalance.target.Target.get
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all
forcebalance.psi4io.RDVR3_Psi4.driver
forcebalance.psi4io.THCDF_Psi4.driver
forcebalance.liquid.Liquid.objective_term
forcebalance.gmxqpio.Monomer_QTPIE.unpack_moments
forcebalance.moments.Moments.unpack_moments
forcebalance.tinkerio.Moments_TINKER.moments_driver

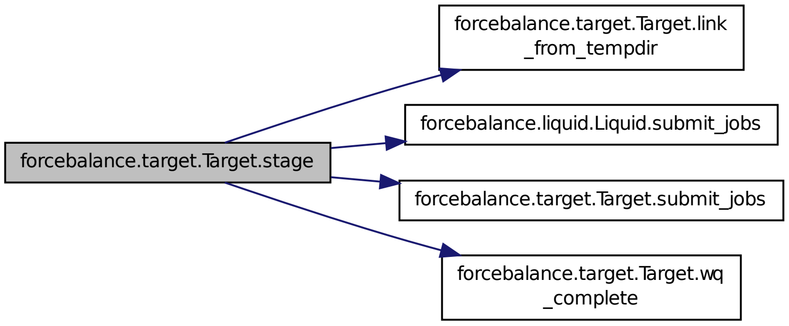
6.46.3.6 def forcebalance.target.Target.stage (*self*, *mvals*, *AGrad* = `False`, *AHess* = `False`, *customdir* = `None`)

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:


forcebalance.target.Target.stage
forcebalance.target.Target.link_from_tempdir
forcebalance.liquid.Liquid.submit_jobs
forcebalance.target.Target.submit_jobs
forcebalance.target.Target.wq_complete

6.46.3.7 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.

6.46.4 Member Data Documentation

6.46.4.1 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 134 of file target.py.

6.46.4.2 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

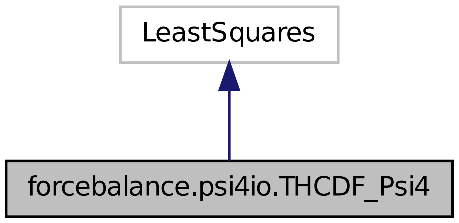
Definition at line 132 of file target.py.

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

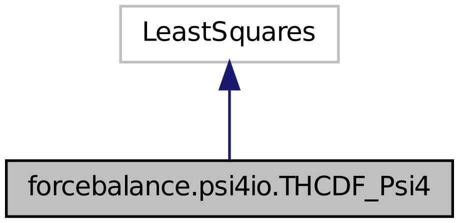
- target.py

6.47 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__**
- 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**

6.47.1 Detailed Description

Definition at line 94 of file psi4io.py.

6.47.2 Member Data Documentation

6.47.2.1 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 235 of file psi4io.py.

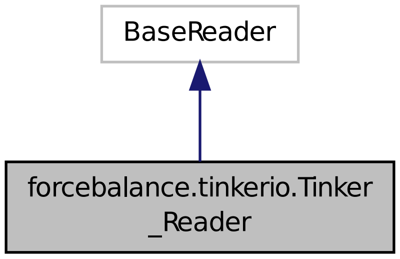
The documentation for this class was generated from the following file:

- psi4io.py

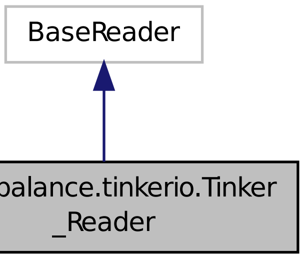
6.48 forcebalance.tinkerio.Tinker_Reader Class Reference

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**

6.48.1 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 66 of file tinkerio.py.

6.48.2 Member Function Documentation

6.48.2.1 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
```

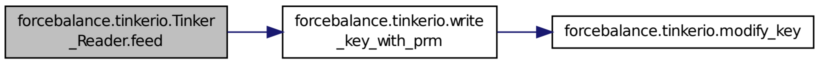
The '#PARM 4' has no effect on TINKER but it indicates that we are tuning the fourth field on the line (the 0.910 value).

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.
```

Definition at line 109 of file tinkerio.py.

Here is the call graph for this function:


forcebalance.tinkerio.Tinker_Reader.feed
forcebalance.tinkerio.write_key_with_prm
forcebalance.tinkerio.modify_key

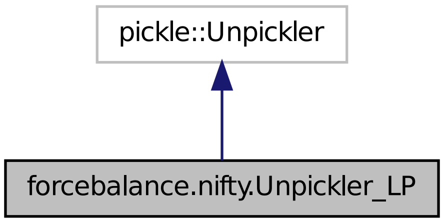
The documentation for this class was generated from the following file:

- tinkerio.py

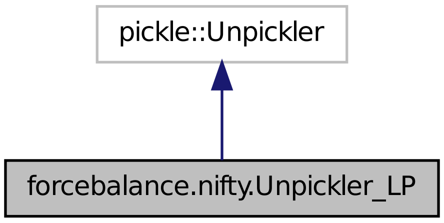
6.49 forcebalance.nifty.Unpickler_LP Class Reference

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Inheritance diagram for forcebalance.nifty.Unpickler_LP:



Collaboration diagram for forcebalance.nifty.Unpickler_LP:



Public Member Functions

- def **__init__**

6.49.1 Detailed Description

A subclass of the python Unpickler that implements unpickling of _ElementTree types.

Definition at line 407 of file nifty.py.

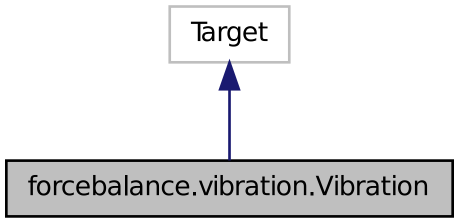
The documentation for this class was generated from the following file:

- nifty.py

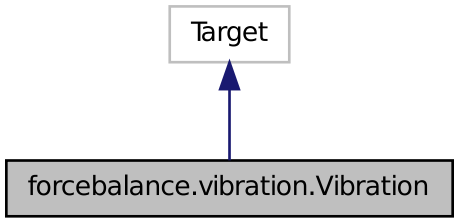
6.50 forcebalance.vibration.Vibration Class Reference

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**

6.50.1 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.

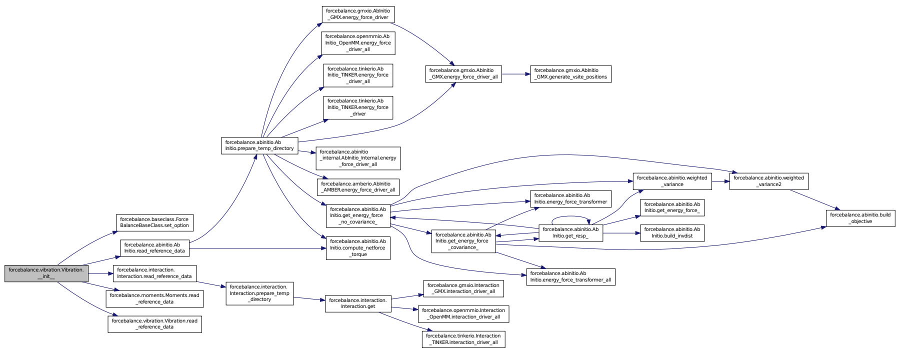
6.50.2 Constructor & Destructor Documentation

6.50.2.1 def forcebalance.vibration.Vibration.__init__ (*self, options, tgt_opts, forcefield*)

Initialization.

Definition at line 32 of file vibration.py.

Here is the call graph for this function:


forcebalance.vibration.Vibration.__init__
forcebalance.baseclass.ForceBalanceBaseClass.set_option
forcebalance.abinitio.AbInitio.read_reference_data
forcebalance.interaction.Interaction.read_reference_data
forcebalance.moments.Moments.read_reference_data
forcebalance.vibration.Vibration.read_reference_data
forcebalance.abinitio.AbInitio.prepare_temp_directory
forcebalance.gmxio.AbInitio_GMX.energy_force_driver
forcebalance.openmmio.AbInitio_OpenMM.energy_force_driver_all
forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver_all
forcebalance.tinkerio.AbInitio_TINKER.energy_force_driver
forcebalance.gmxio.AbInitio_GMX.energy_force_driver_all
forcebalance.gmxio.AbInitio_GMX.generate_vsite_positions
forcebalance.abinitio_internal.AbInitio_Internal.energy_force_driver_all
forcebalance.amberio.AbInitio_AMBER.energy_force_driver_all
forcebalance.abinitio.AbInitio.get_energy_force_no_covariance_
forcebalance.abinitio.AbInitio.compute_netforce_torque
forcebalance.abinitio.AbInitio.energy_force_transformer
forcebalance.abinitio.AbInitio.get_energy_force_covariance_
forcebalance.abinitio.AbInitio.get_resp_
forcebalance.abinitio.weighted_variance
forcebalance.abinitio.weighted_variance2
forcebalance.abinitio.AbInitio.get_energy_force_
forcebalance.abinitio.AbInitio.build_invdist
forcebalance.abinitio.build_objective
forcebalance.abinitio.AbInitio.energy_force_transformer_all
forcebalance.interaction.Interaction.prepare_temp_directory
forcebalance.interaction.Interaction.get
forcebalance.gmxio.Interaction_GMX.interaction_driver_all
forcebalance.openmmio.Interaction_OpenMM.interaction_driver_all
forcebalance.tinkerio.Interaction_TINKER.interaction_driver_all

6.50.3 Member Function Documentation

6.50.3.1 def forcebalance.vibration.Vibration.get (*self, mvals, AGrad* = `False`, *AHess* = `False`)

Evaluate objective function.

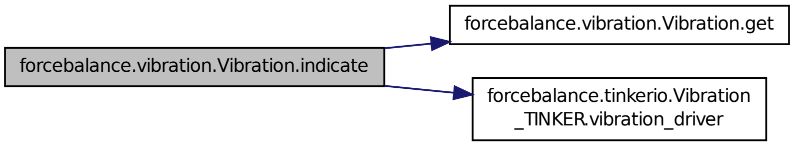
Definition at line 108 of file vibration.py.

6.50.3.2 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:


forcebalance.vibration.Vibration.indicate
forcebalance.vibration.Vibration.get
forcebalance.tinkerio.Vibration_TINKER.vibration_driver

6.50.3.3 def forcebalance.vibration.Vibration.read_reference_data (*self*)

Read the reference vibrational data from a file.

Definition at line 54 of file vibration.py.

6.50.4 Member Data Documentation

6.50.4.1 forcebalance.vibration.Vibration.vfnm

The vdata.txt file that contains the vibrations.

Definition at line 46 of file vibration.py.

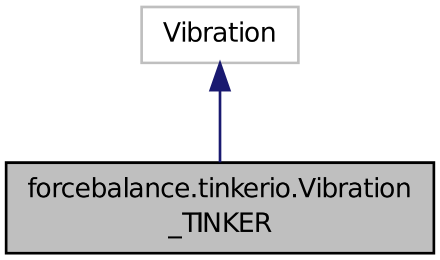
The documentation for this class was generated from the following file:

- vibration.py

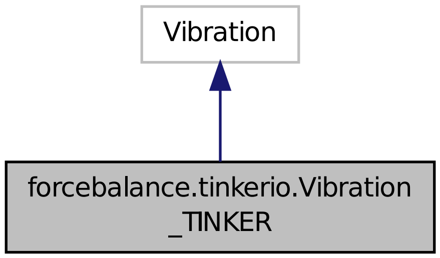
6.51 forcebalance.tinkerio.Vibration_TINKER Class Reference

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**

6.51.1 Detailed Description

Subclass of Target for vibrational frequency matching using TINKER.

Provides optimized geometry, vibrational frequencies (in cm-1), and eigenvectors.

Definition at line 348 of file tinkerio.py.

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

- tinkerio.py

Index