pyMDMix Documentation

Release 0.1.1

Daniel Alvarez-Garcia

March 11, 2014

CONTENTS

Contents:

CHAPTER

ONE

PROJECT MODULE DOCUMENTATION

1.1 Project class

class `pyMDMix.Project.`**`Project`**(*name='mdmix_project'*, *amberPDB=None*, *amberOFF=None*, *projectFilePath=None*, *unitName=None*, *extraResList=[ ]*, *restrainMask='auto'*, *alignMask='auto'*, *extraFF=[ ]*, ***kwargs*)

Class to hold all project related options

`__init__`(*name='mdmix_project'*, *amberPDB=None*, *amberOFF=None*, *projectFilePath=None*, *unitName=None*, *extraResList=[ ]*, *restrainMask='auto'*, *alignMask='auto'*, *extraFF=[ ]*, ***kwargs*)

A project can be initialized in four ways::

- Empty: The user may call later `setOFF()` or `createOFFFromPDB()` to set expected Amber object file and system unit name needed by the project to proceed.
- From existing project file: *projectFilePath* can point to a folder containing a project file (*.mproj) to be loaded. All information will be loaded from existing sources.
- With a PDB file: *amberPDB* can point to a PDB file path with a system ready to be converted to Object File (main input to Project). If the PDB contains non-standard residues, the user can provide forcefield modification files or parameters in *extraFF* list.
- From an Amber Object File (OFF): Pass in *amberOFF* a path to an amber object file containing the system to simulate. Extra forcefield parameters and modifications can be passed in *extraFF* list.

`addNewReplica`(*replica*, *updateReplica=True*)

Add new replica to current project. If replica.name exists in current project, a warning will be raised and nothing done.

If the new replica does not have a name, an automatic one will be assigned with format SOLVENT_NUM depending on the number of replicas per solvent already present in current project (e.g. ETA_0 if its the first replica with ETA as solvent; MAM_3 if it's the fourth replica with this same solvent).

If *updateReplica* is **True**, replica restrain mask, align mask, forcefield information extra residues and reference PDB will be replaced with current project information.

Parameters

- **replica** (`Replica`) – Replica to add to current project. It can be created later.
- **updateReplica** (*bool*) – Update replica information with project details.

`amberOFF` **= None**

Amber object file path with the system to be studied

`amberPDB` **= None**
PDB file to be used in the preparation of an Amber Object File

`createGroup``(`*groupname*, *replicanames*`)`
Create a group of replicas for joint analysis

Parameters

- **groupname** (*str*) – Name to identify the group
- **replicanames** (*list*) – Replica names to add to group

`createNewReplicas``()`
Create folder structure and MD input for replicas not already created. If the project folder is not created, it will also be created.

`createOFFFromPDB``(`*amberPDB*, ***kwargs*`)`
Create an amber object file from the pdb given. This PDB file should be prepared to be read by tLeap (correct residue and atom namings) Will automatically try to cap the pdb at all terminus and build SS bonds.

This method will save and return the object file and the unit name containing the system. Call `setOFF()` to assign them to the project.

Parameters **amberPDB** (*str*) – File path to PDB to be saved as object file

Kwargs Check `cleanPDB()` method for extra parameters that are accepted to control automatic cleaning of the PDB.

Returns Object file path recently created and the unit name containing the system.

Return type (str, str)

`createProject``()`
Write directory tree and project files. This method will call createProjectFolder and createProjectFiles.

`createProjectFiles``()`
Save reference PDB, PRMTOP and PRMCRD from Project Object File.

`createProjectFolder``()`
Create folder structure for current project name and update paths

`extraFF` **= None**
Forfecield parameters or frcmods to define the system

`getGroup``(`*groupname*`)`
Get a list with replicas belonging to the group *groupname*

Parameters **groupname** (*str*) – Name of the group to retrieve

Returns List with `Replica` instances or False if group does not exists

`getSolventList``()`
Return list of solvents used in current project

`load``(`*projfile=None*`)`
Load existing project from pickled file

`projFileName` **= None**
Name of the file that will be generated to save all project information

`projName` **= None**
Name of the project

`removeGroup`(*groupname*)
Remove group. :arg str groupname: group name to be removed from project

`removeReplica`(*replicaname*)
Remove replica from current project

Parameters **replica** (*str*) – replica name to be removed from project

Returns *True* if replica was removed. *False* if the name was not found.

`setOFF`(*offpath*, *unitname=False*)
Set current project amber object file containing the system under study.

Parameters

- **offpath** (*str*) – Valid file path to an amber object file.
- **unitname** (*str*) – Unit name inside the file that contains the system. If not given, will automatically use the first unit found in the file.

`unitName` = **None**
Name of the unit inside the AmberOFF containing the system in study

`updatePath`()
Update main project folder path with current working directory. All project replica's paths will be also updated.

This method to work requires that the expected project file (`Project.projFileName`) is placed inside the current working directory.

`updateReplica`(*replica*)
Update replica. When a replica is modified, it is recomended to run this method to update project file information. :arg replica: replica instance to be updated in current project :type replica: `Replica`

`updateReplicaPaths`()
Update replica paths with current project main path information

`write`()
Save object **__dict__** to pickled file.

CHAPTER

TWO

REPLICAS MODULE DOCUMENTATION

This module contains the main class `Replica` for storage and manipulation of a single simulation run.

2.1 Aid to developers

2.1.1 Replica flexible configuration

`Replica` objects can be configured by mean of their constructor method. Arguments not present in at construction time, will take default values from default settings and user settings **replica-settings.cfg** files.

```
>>>from Replicas import Replica
>>>replica = Replica('customreplica', nanos=40, temp=298)
>>>print replica.nanos
40
>>>print replica.temp
298
>>>defreplica = Replica()
>>>print defreplica.nanos
20
>>>print defreplica.temp
300
```

temp and *nanos* attributes where assigned from user configuration file which should be at user's home directory $HOME/.mdmix/replica-settings.cfg. Values not explicitely assigned there, will be taken from default configurations in package installation directory.

This system for building instances permits the developer to modify/add/remove attributes to the instance without modifying any code. For instance, if a new pair *int-myattr=200* is written in user's replica configuration file, default replicas will also have that new attribute.

2.2 Replica class

class `pyMDMix.Replicas.`**`Replica`**(*solvent=None*, *name=None*, *pdb=None*, *crd=None*, *top=None*, *path=None*, *extraResidues=*[ ], *restrMask='‘*, *alignMask='‘*, *refPdb=None*, ***kwargs*)

Class to contain an independent simulation run (a Replica). Create folders, input files and control completeness of the different steps.

`__init__`(*solvent=None*, *name=None*, *pdb=None*, *crd=None*, *top=None*, *path=None*, *extraResidues=*[ ], *restrMask='‘*, *alignMask='‘*, *refPdb=None*, ***kwargs*)

Constructor method for Replica objects.

Parameters

- **name** (*str*) – Replica name. Name should be given now or later with setName(name).
- **solvent** (*str*) – Solvent name. It must exist in Solvent database.
- **pdb** (*str*) – PDB file containing the solvated system. To be created if still non existant.
- **crd** (*str*) – Amber PARMCRD file of the solvated system. To be created if still non existant.
- **top** (*str*) – Amber PRMTOP file. To be created if still non existant.
- **path** (*str*) – Path to replica folder structure. If not given now, can be assigned with `setPath()`.
- **extraResidues** (*list*) – List of non-standard residue names we wish to consider as part of the 'solute' (important with auto mask detection).
- **restrMask** (*str*) – Amber format mask to select residues and atoms to be restrained (if needed). If emtpy and restrains are requested, an automatic mask will be calculated from the pdb
- **alignMask** (*str*) – Amber format mask to select atoms and residues over which trajectory should be aligned. If empty, an automatic mask will be calcualted.
- **refPdb** (*str*) – Path to a PDB file used as reference for trajectory alignment.

Keywords This provides a very flexible attribute assignment system. - Every pair key=value will be assigned as an attribute to current replica. - Pair values not given will take default values from Global Settings or User Settings specifications.

`addFF`(*ffname*)
Add forcefield parameters or FF modification files to be loaded when preparing the system with tLeap.

`attach`(*object*, *attachname*, *desc=''*)
Attach an object to this replica. This will create a pickle of the object with a temporary filename and store the pickle file name with a *name* and *description* in current replica attribute `Replica.attached` dictionary.

Parameters

- **object** (*any*) – Object to pickle and link to the replica
- **attachname** (*str*) – Name to assign to the attachment
- **desc** (*str*) – Description. Optional.

`createFolders`()
Create directory tree for current replica. `path` should have been set with `setPath()`. Copy inside the top/crd/pdb files if given.

Tree structure:

replica.name/ replica.minfolder/ replica.eqfolder/ replica.mdfolder/

`createMDInput`()
Create MD input config files for the program selected (AMBER or NAMD).

`createSystemFromOFF`(*systemoff*, *systemunit*, *prmtop=None*, *prmcrd=None*, *pdb=None*)
Using tLeap, create a AMBER PRMTOP and PRMCRD from the *systemoff* filepath and unitname *systemunit*. Solvent box will be added according to the solvent name used when instantiating the Replica. Files will be saved inside replica folder.

Parameters

- **systemoff** (*str*) – path to Amber Object File with system saved.
- **systemunit** (*str*) – name of the unit saved inside *systemoff* to be prepared.
- **prmtop** (*str*) – name of the PARMTOP file to be saved. Default name will be constructed from *systemunit* and name of replica. E.g.: MYSYS_ETA0.prmtop
- **prmcrd** (*str*) – name of the PARMCRD file to be saved. Default name will be constructed as for PARMTOP with extension .parmcrd
- **pdb** (*str*) – name of the PDB file to be saved from the PARMTOP and PARMCRD files. Default name constructed as the otehrs with extension .pdb

`dettach`(*attachname*)
Remove attachement with name **attachname**

`folderscreated`()
Return **True** if replica directory structure is created.

`getAttached`(*attachname*)
Load attached object. See ::method::*attach* for more info.

Returns Unpickled object.

`go`()
Move to replica folder if created

`importData`(***kwargs*)
Import existing data into current replica. Useful when analyzing data from external simulations not run under pyMDMix.

Keywords Give key=value pairs to be imported where: - **value**: absolute path of existing folder containing data to link or a existing file. - **key**: repica folder or file where to link data to:

FILES:

- *pdb*: System PDB
- *top*: System Amber Topology
- *crd*: System Amber Coordinates
- *solvent*: Simulated solvent

FOLDERS

- *mdfolder*: Production trajectory and output files
- *eqfolder*: Equilibration folder
- *alignfolder*: Aligned trajectory folder
- *densityfolder*: Containing density grids
- *energyfolder*: Contraining energy converted grids

All keys are optional. Only keys assigned will be imported.

Example::~

```
>>> replica = Replica('ETA', name='test')
>>> replica.importData(pdb='/oldfolder/system.pdb', crd='/oldpath/system.crd', top
>>>                      mdfolder='/oldpath/production', eqfolder='/oldpath/equilibr
```

E.g. /oldfolder/production content will be linked into `pyMDMix.Replicas.Replica.mdfolder` folder.

iscreated()
Return **True** if replica folder and MD inputs have been written

mdinputwritten()
Return **True** if replica MD input files are writen.

setName(*name*)
Set replica name. Adapt logger.

setNanos(*nanos*)
Change number of nanoseconds for current replica

setOutFileTemplate(*outfiletemplate*)
Set/Modify output filename template for current replica. All filename templates must include {nano} and {extension}. E.g.: md{nano}.{extension}

setPath(*path*, *update=True*)
Set replica path. If update is True, update subfolder paths.

updateFromSettings()
Auxiliary function to update object with attributes from configuration files.

updatePaths()
Update replicaPaths using `path` as base path

2.3 Examples

2.3.1 Importing existing data

In this example we will create an empty replica folder and add existing sources from an imaginary previous simulation. It was run with ethanol mixture (named *ETA*) for 40ns.

```
>>> previousdata = {'pdb':'/some/path/system.pdb', 'crd':'/some/path/system.crd','top':'/some/path/sy
>>> from Replicas import Replica
>>> replica = Replica('mynewreplica', nanos=40, solvent='ETA')
>>> replica.createFolders()
>>> replica.importData(**previousdata) # This will link all existing files inside the created folder
```

CHAPTER

THREE

SOLVENTS MODULE DOCUMENTATION

This module provides main `Solvent` object class and a `SolventManager` class to create/remove solvents from the solvent database.

The easiest way to create a new solvent is through a configuration file where all parameters can be assigned.

Second way is to instantiate the solvent directly giving all required parameters to the constructor method.

3.1 Instantiating and working with Solvent objects

Solvent objects might be instantiated giving all required parameters to the constructor. Here I exemplify it's usage and basic attributes:

```
>>> import pyMDMix.tools as T
>>> import pyMDMix.Solvents as S
>>>
>>> off_file = T.testRoot('ETAWAT20.off')    # Amber object file with box unit
>>> name = 'ETA'
>>> probesmap = {'OH':'ETA@O1', 'CT':'ETA@C2'}    # Will link probe name OH with residue name ETA ato
>>> typesmap = {'OH':'Don,Acc', 'CT':'Hyd'}      # Link probe OH with types Donor and Acceptor. Link 
>>>
>>> boxunit = 'ETAWAT20' # As named inside the off file
>>> info = 'Test direct instantiation of a Solvent object'
>>>
>>> # Create instance
>>> solv = S.Solvent(name=name, info=info, off=off_file, probesmap=probesmap, typesmap=typesmap, boxu
>>> print solv
SOLVENT: ETA
INFO: Test direct instantiation of a Solvent object
BOXUNIT: ETAWAT20
>>> print solv.probes # print configured probes
[CT, OH]
>>> print solv.types  # set object
set(['Acc', 'Don', 'Hyd'])
>>>
>>> # Calculate the probability of finding atom O1 of residue ETA in a grid voxel of 0.5x0.5x0.5 Ang
>>> print solv.getProbability('ETA','O1',voxel=0.5**3)
0.0005320194076762995
```

3.2 Adding new solvents to databases

The Solvent object must be configured using `SolventManager.createSolvent()` method giving a configuration file as argument. Here is an exaple of valid configuraiton file with all available options commented:

```
[GENERAL]
# name to identify the mixture (ex: ION)
name = ETA
# Any string to describe the box
info = Ethanol 20%% mixture
# path to off file containing the leap units
# It should contain all parameters
objectfile = ETAWAT20.off
# If the box contains waters, the name of the model (TIP3P, TIP4P, SCP..)
watermodel = TIP3P
# Name of the Leap box unit in object file(ex: IONWAT20)
boxunit = ETAWAT20

[PROBES]
# OPTIONAL SECTION
# map probe names with residue@atoms (ie. NEG=COO@O1,O2)
# probe names must be unique
WAT=WAT@O
CT=ETA@C1
OH=ETA@O1

[TYPES]
# OPTIONAL
# Assign chemical types to the probes in previous section
# Example: OH=Donor,Acceptor
OH=Don,Acc
CT=Hyd
WAT=Wat
```

Configuring and adding a new solvent into the default database:

```
>>> import tools as T
>>> from Solvents import SolventManager
>>> SM = SolventManager()
>>>
>>> # Read configuration and create object.
>>> # Object file path in the configuration file must be correct or
>>> # errors will arise
>>> configfile = T.testRoot('solvent_template.cfg')
>>> print configfile
/Users/dalvarez/Dropbox/WORK/pyMDMix/pyMDMix/data/test/solvent_template.cfg
>>> newsolvent = SM.createSolvent(configfile)
>>> print newsolvent
SOLVENT: ETA
INFO: Ethanol 20% mixture
BOXUNIT: ETAWAT20
>>>
>>> # Add to the database
>>> SM.saveSolvent(newsolvent)
ETA saved to database /Users/dalvarez/Dropbox/WORK/pyMDMix/pyMDMix/data/solventlib/SOLVENTS.db
>>> SM.listSolvents()
['PYZ', 'ISX', 'TFE', 'CLE', 'MSU', 'IMZ', 'ANT', 'ION', 'ETA', 'MOH', 'ISO5', 'ETAA', 'ISO', 'WAT',
```

Configure and save object in a specific, empty database:

```
>>> # This action will copy all solvents in the default db to the new db and add the new solvent
>>> customdb = 'mycustomdb.db'
>>> SM.saveSolvent(newsolvent, db=customdb)
>>> SM.listSolvents(customdb)
['PYZ', 'ISX', 'TFE', 'CLE', 'MSU', 'IMZ', 'ANT', 'ION', 'ETA', 'MOH', 'ISO5', 'ETAA', 'ISO', 'WAT',
>>> # Optionally, an empty database can be created and only add the new solvent
>>> SM.saveSolvent(newsolvent, db='otherdb.db', createEmpty=True)
>>> SM.printSolvents('otherdb.db')
['ETA']
```

3.3 Solvent class

class `pyMDMix.Solvents.`**`Solvent`**(*name*, *info*, *offpath*, *boxunit*, *probesmap*, *typesmap*, *frcmodpaths=[ ]*, *watermodel='TIP3P'*, **args*, ***kwargs*)

Solvent class for storing information on solvent mixtures for simulations

`__init__`(*name*, *info*, *offpath*, *boxunit*, *probesmap*, *typesmap*, *frcmodpaths=[ ]*, *watermodel='TIP3P'*, **args*, ***kwargs*)

Constructor of a Solvent object. It expects some mandatory fields and some optional extra fields that can be assigned through kwargs.

Parameters

- **name** (*str*) – Name identifying the current solvent mixture.
- **info** (*str*) – Some string describing the solvent mixture. Will be used for printing the solvent.
- **off** (*str*) – Filename that must exist. Will be used to fetch all information about the mixture and also to later solvate the systems for simulation. Make sure this file is correct and the units correctly working when setting up systems with it.
- **boxunit** (*str*) – String specifying the name of the Leap unit containing the mixture inside the objectfile. Mandatory, specially important when more than 1 units are saved inside same object file.
- **probesmap** (*dict*) – Map probe names to residues and atom names in the solvent box. Check documentation at the website or the documents for more information.
- **typesmap** (*dict*) – Dictionary mapping chemical types to the probes in *probesmap*.
- **watermodel** (*str*) – If the solvent box contains waters, specify the water model used. Example; TIP3P, TIP4P... By default, it will be assigned to `S.DEF_WATER_BOX`

Keywords key=value pairs that will be set as Solvent attributes. Useful for adding extra information in the solvent instance for easy access from custom functions.

probemap and *typesmap* Examples:

```
>>> probemap = {'OH':'ETA@O1'} # Identify atom O1 of residue ETA as probe OH.
>>> typesmap = {'OH':'Don,Acc'} # Assign probe OH the chemical types Don and Acc (for Donor
```

kwargs assignment example:

```
>>> solvent = Solvent(name='mysolvent',info='custom solvent',off='path/to/objectfile.off',.
>>> print solvent.myspecialattr
300
```

getProbability(*res*, *atoms*, *voxel=None*)
Obtain expected probability for all the atoms to fall into the voxel volume.

Parameters

- **res** (*str*) – Residue name.
- **atoms** (*list*) – Atom name list
- **voxel** (*float*) – Volume of the voxel. If not given, it is automatically calculated from defaults.

Returns Probability of finding any of the atoms in a voxel.

Type float

getProbeByName(*name*)
Returns `Probe` instance with name *name*. **False** otherwise.

getProbeProbability(*probename*)
Return probe probability

getProbesByType(*type*)
Returns a list of `Probe` instances that match type *type*.

getResidue(*resname*)
Return `Residue` object with name *resname*

isIonic()
Check wether the solvent box contains charged residues

writeOff(*name*)
Write object file of current solvent to filname *name*

3.4 SolventManager class

class pyMDMix.Solvents.**SolventManager**
Class to manage solvent creation/removal and database manipulation

createSolvent(*configfile*)
Create a Solvent isntance from the information in *configfile* configuration file. Examples of this file are available at template folder `T.templatesRoot()`

Parameters **configfile** (*str*) – Filename with solvent configuration file

Return solvent Return a Solvent object configured.

Return type `Solvent`

getDatabase(*db=None*)
Open the database / unpickle it.

getSolvent(*name*, *db=None*)
Fetch solvent from the database by name.

Parameters

- **name** (*str*) – Solvent name.
- **db** (*str*) – Database path. If None, automatically detect.

Return Solvent object from the database

Return type `Solvent` instance or False if not found.

`listSolvents`(*db=None*)

Fetch solvent names from the database.

Parameters **db** (*str*) – Database path. If None, automatically detect.

Return Name list

Return type list

`printSolvents`(*db=None*)

Like list solvents but will print to screen information about the solvents.

`removeSolvent`(*solvName*, *db=None*)

Remove solvent from database Same as saveSolvent, `db` will be chosen automatically if None.

Parameters

- **solvName** (*str*) – Solvent name.
- **db** (*str*) – Database path. If None, automatically detect.

Raises SolventManagerError if *db* does not contain *solvName*.

`saveSolvent`(*solvent*, *db=None*, *createEmpty=False*)

Save a Solvent isntance in the database `db` or default DB locations. Selection of database is done in `self.__getDatabase()`.

Parameters

- **solvent** (`Solvent`) – Solvent object to save.
- **db** (*str*) – Database where to save the solvent. If None, default ones will be used (package DB if the user can write there, or user DB otherwise).
- **createEmpty** (*bool*) – If new database, create empty. If False, copy data from package DB to the new DB.

CHAPTER

FOUR

CONTAINERS MODULE DOCUMENTATION

class `pyMDMix.containers.`**`Atom`**(*id*, *name*, *type*, *element*, *charge*, **args*, ***kwargs*)

Simple container for atomic information gathered in the OFF file: name, type, element, charge

`charge` **= None**

Partial charge

`element` **= None**

Element. Integer.

`id` **= None**

Integer. Atom ID.

`name` **= None**

Atom name

`type` **= None**

Atom AMBER TYPE

class `pyMDMix.containers.`**`Probe`**(*name*, *residue*, *atoms*, *type*, *probability*)

Container for probe information. This object will store information about a particular probe linking atom names, residues and chemical types with probabilities. Will also contain a mask for the residue.

`atoms` **= None**

Atom name list

`name` **= None**

Name of the probe as given in `Solvent.probesmap`

`p` **= None**

Probability of probe to be found in grid voxel volume

`residue` **= None**

`Residue` instance with corresponding residue information

`type` **= None**

Chemical type

class `pyMDMix.containers.`**`Residue`**(*name*, *atoms*, *connectivity*, *xyz*, **args*, ***kwargs*)

Simple container for whole residue unit information gathered in the OFF file. Basically to sotre atomic information and possibly masks for later quick identify atom positions.

`atids` **= None**

Dictionary mapping atom ids to `Atom` instances

`atnames` **= None**

Dictionary mapping atom names to `Atom` instances

`atoms` = None
List of `Atom` instances that belong to residue

`charge` = None
Total charge of the residue

`connectivity` = None
Tuple with connectivity information

`name` = None
Name of the residue

`xyz` = None
XYZ coordinates in a numpy array Nx3

CHAPTER

FIVE

OBJECT FILE MANAGEMENT MODULE DOCUMENTATION

This module provides a reader for Amber OFF file format.

Example::

```
>>> import os.path as osp
>>> import pyMDMix.tools as T
>>> import pyMDMix.OFFManager as O
>>>
>>> f_in = osp.join(T.testRoot(), 'ETAWAT20.off')
>>> m = O.OFFManager(offFile=f_in)
>>>
>>> print m.getUnits() # Get unit names present in the OFF file
['ETA','ETAWAT20','WAT']
>>> print m.getResidueList('ETAWAT20', unique=True) # Get residues inside unit 'ETAWAT20'
['WAT','ETA']
>>> print m.getVolume('ETAWAT20') # Volume of the box
7988.43038
>>> print m.getNumRes('ETAWAT20', 'ETA') # Number of 'ETA' residues inside 'ETAWAT20' unit.
17
```

class `pyMDMix.OFFManager.`**`OFFManager`**(*offFile=None*, *offString=None*, **args*, ***kwargs*)
Manage Amber OFF file types. Only for reading.

`getAtoms`(*unit*, *skipH=False*)
Fetch atomic information for the unit selected. Will return a dictionary with atom names and types.

Parameters

- **unit** (*str*) – Unit to search.
- **skipH** (*bool*) – If atom is Hydrogen, skip it.

Returns List of `Atom` instances containing id, name, atomtype, element and charge information.

Return type list

`getBoxDimensions`(*unit*)
Get box dimension information from the object file for `self.boxunit`

`getConnectivity`(*unit*)
Fetch connectivity table for unit selected. This is section !entry.UNIT.unit.connectivity in off file.

Parameters **unit** (*str*) – Unit name

Returns List with bonded pair indexes verbosely. Example: ((1,2),(1,3),(3,1),(2,1)...)

Return type list

getCoords(*unit*)
Fetch positions information for unit selected. This is section !entry.UNIT.unit.positions in off file.

Parameters **unit** (*str*) – Unit name

Returns Coordinates of unit atoms.

Return type `numpy.ndarray` of floats with size *Nx3*

getNumRes(*unit*, *residue*)
Count the number of residues with name *residue* inside unit *unit*

getResidue(*res*, *skipH=False*)
Fetch residue in off and return a `Residue` instance aontaining also atomic information.

Parameters

- **unit** (*str*) – Residue name which should correspond to a valid unit in off file.
- **skipH** (*bool*) – Skip hydrogen atom information. Default:False.

Returns Residue instance info.

Return type `Residue` or **False** if unit not found.

getResidueList(*unit*, *unique=True*)
Get a list of residue names for the *unit* chosen.

Parameters

- **unit** (*str*) – Unitname to search.
- **unique** (*bool*) – If True, return a list with unique names. If False, the complete list of names will be returned.

Returns List with residuenames inside unit *unit*.

getUnits()
Return the list of units int he object file.

getVolume(*unit*)
Get volume information from the object file for `self.boxunit`

hasUnit(*unitname*)
Return True if the OFF file has the unit with name 'unitname'.

isParameter(*unit*)
Check if unit *unit* is a parameter unit inside OFF.

Parameters **unit** (*str*) – Name of the unit

Returns True if its a parameter unit. False otherwise.

readOffSection(*unit*, *section*, *with_header=False*)
Parse the object file and read a whole section for the unit selected.

Parameters

- **unit** (*str*) – Unit name to search.
- **section** (*str*) – OFF file section name. Example: *residues* section will correspond to "!entry.UNITNAME.unit.residues .." part of the file.
- **with_header** (*bool*) – If True, return output with heading line of the section.

Returns Content of the section unitl next '!entry' is found. Returned with our without the heading line depending on the value of *with_header*

Return type list of strings

class `pyMDMix.OFFManager.`**`Test`**(*methodName='runTest'*)

Test

`test_OFFManager`()

OFFManager test

CHAPTER

SIX

SETTING GENERAL OPTIONS AND ATTRIBUTES THROUGH CONFIGURATION FILES

Several classes automatically take arguments from default configuration files distributed along the package. Once pyMDMix is started for the first time, it will also make a copy of these configuration files inside the user's home directory for easy modification. If any parameter is modified by the user, it will have higher priority and the default one will be ignored. For restoring initial file, just remove it from the user directory.

Mainly, three configuration files govern the program:

- General configuration (**settings.cfg**)
- Replica configuration (**replica-settings.cfg**)
- Project configuration (**project-settings.cfg**)

6.1 General configuration

This is the default file for configuring general and project options in pyMDMix. It can be found at the package installation directory ($INSTALLDIR) under **$INSTALLDIR/data/defaults/settings.cfg** or at user's home directory **./mdmix/settings.cfg**

```
[MD]
list-AVAIL_MDPROG = AMBER, NAMD # IMPLEMENTED SIMULATION PROGRAMS
float-AMBER_SOLVATE_BUFFER = 13 # Buffer for solvateOct command in tLeap
DEF_AMBER_WATBOX = TIP3P        # Default water model to use
DEF_MDPROGRAM = AMBER                # From implemented programs, default to use
int-DEF_TRAJFREQUENCY = 500                         # Trajectory ritting frequency  = 1000 snapshots
int-DEF_MINSTEPS = 5000                                # Number of minimization steps to run
int-DEF_HEATING_STEPS_PER_FILE = 100000         # Heating steps. 100.000 steps = 200ps
int-DEF_HEATING_TEMP_INI = 100                         # Start heating at 100 K
int-DEF_NPT_EQUILIBRATION_NSTEPS = 500000         # 1ns equilibration at NPT
int-DEF_NVT_PRODUCTION_NSTEPS = 500000                 # 1ns production files
int-DEF_NAMD_HEATING_TOTALSTEPS = 500000         # 1ns equilibration time to increase temperature fro
int-DEF_AMBER_NETCDF = 1                            # Write trajectory in NETCDF format by default
list-DEF_FF = leaprc.ff99SB, leaprc.gaff        # Default forcefield files to load when opening tLeap

[GENERAL]
## The following options are only ckecked for their type
## use type-name if you want to enforce type conversion on a parameter
## example:
## int-param1 = 10 creates a variable param1 of type int with value 10
## By contrast, param1 = 10 gives a variable param1 of type str with value '10'
```

```
int-testparam = 42        ## used for test code
list-GRIDTYPES = MDMIX_DENS,MDMIX_CORR,MDMIX_RAW,MDMIX_OTHER,MDMIX_UNK, MDMIX_PART_DENS, MDMIX_RAW_AV
AVGOUTPATH = PROBE_AVG
AVGOUTPREFIX = avg_
float-GRID_SPACING = 0.5
DEBUG=0                  # If zero, no extra debug information will be printed. Put 1 for extra info.
```

6.2 Replica configuration

```
[GENERAL]
## The following options are only ckecked for their type
## use type-name if you want to enforce type conversion on a parameter
## example:
## int-param1 = 10 creates a variable param1 of type int with value 10
## By contrast, param1 = 10 gives a variable param1 of type str with value '10'
## Possible types: int, float, bool, list.
## list type will chop the string by commas. Eg. list-ff=a,b will become a list ff=['a','b']

# Set simulation options
netcdf = 1                  # 1 (write trajectory in nc format) or 0 (write in ascii format)
restrMode = FREE            # Restraining scheme: FREE, HA (heavy atoms) or BB (backbone only)
float-restrForce = 0.0      # Restraining force if applicable. Default 0 kcal/mol.A^2
int-nanos = 20              # Production length in nanoseconds. Default: 20ns
float-temp = 300            # Simulation temperature. Default = 300K
mdProgram = AMBER           # Default simulation program. Options: AMBER or NAMD currently
int-trajfrequency = 500     # Trajectory writing frequency  = 1000 snapshots per nanosecond = int-pro
int-minsteps = 5000                                 # Number of minimization steps to run
int-heating_steps = 100000                      # Heating steps for each file. 100.000 steps = 200ps
float-parm_heating_tempi = 100                          # Start heating at 100 K
int-npt_eq_steps = 500000                       # 1ns equilibration at NPT
int-nvt_prod_steps = 500000                     # 1ns production files
int-namd_heating_steps = 500000                 # 1ns equilibration total time to increase temperatur
list-FF = leaprc.ff99SB, leaprc.gaff            # Default forcefield files to load when opening tLeap
float-amber_solvate_buffer = 14                 # Buffer for solvateOct command in tLeap

# Set filepaths
mdfolder = md               # Name for the production folder
eqfolder = eq               # Name for equilibration folder
minfolder = min             # Name for minimization folder
alignfolder = align         # Name for folder containing aligned trajectory
energyfolder = egrids       # Name for folder containing energy grids
densityfolder = dgrids      # Folder containing density/occupancy grids

# Next option is a string to be converted to trajectory file names in production and equilibration f
# Always include {nano} and {extension} keywords.
# Example:    md{nano}.{extension}    (DEFAULT)
# Will be:    md1.nc  if netcdf used and 1st nanosecond of production.
outfiletemplate = md{nano}.{extension}
```

#Settings Module #=============== # #.. automodule:: Settings # :members:

CHAPTER

SEVEN

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

PYTHON MODULE INDEX