molecuPy Documentation

Release 1.1.0

Sam Ireland

January 29, 2017

Contents

molecuPy is a Python parser for Protein Data Bank (PDB) files. It provides utilities for reading and analysing the structural data contained therein.

CHAPTER 1

Example

```
>>> import molecupy
>>> pdb = molecupy.get_pdb_remotely("1LOL")
>>> pdb.title()
'CRYSTAL STRUCTURE OF OROTIDINE MONOPHOSPHATE DECARBOXYLASE COMPLEX WITH XMP'
>>> pdb.model()
<Model (3431 atoms)>
>>> pdb.model().get_chain_by_id("A").mass()
20630.8656
```

CHAPTER 2

Table of Contents

2.1 Installing

2.1.1 pip

molecuPy can be installed using pip:

```
$ pip install molecupy
```

molecuPy is written for Python 3. If the above installation fails, it may be that your system uses `pip` for the Python 2 version - if so, try:

```
$ pip3 install molecupy
```

2.1.2 Requirements

molecuPy requires the Python libraries requests and OmniCanvas. These will be installed automatically if molecuPy is installed with pip.

Otherwise molecuPy has no external dependencies, and is pure Python.

2.2 Overview

2.2.1 Creating Pdb objects

There are two main ways to create a Pdb object from a PDB file. The first is from a local PDB file:

```
>>> import molecupy
>>> pdb = molecupy.get_pdb_from_file("path/to/file.pdb")
```

This is the quickest way, though it is not always convenient to store PDB files locally. The second way is to fetch the PDB file over the internet:

```
>>> import molecupy
>>> pdb = molecup.get_pdb_remotely("1LOL")
```

This takes longer, but it means you can access any published PDB file without needing to manually download them first.

However the text of the PDB file is obtained, the process of parsing it is always the same:

1. First a *PdbFile* object is created, which is a representation of the file itself. This is essentially a list of records, with methods for getting records of a certain name.

2. This is used to make a *PdbDataFile* object. This is the object which extracts the data from the file, and is essentially an unstructured list of values.

3. This is used to make a *Pdb* object, by using the values in the data file to create a user-friendly handle to the information. This is the object returned by the above two methods.

2.2.2 Accessing Pdb properties

Aside from structural information, PDB files also contain many other pieces of information about the file, such as its title, experimental techniques used to create it, publication information etc.

```
>>> pdb.pdb_code()
'1LOL'
>>> pdb.deposition_date()
datetime.date(2002, 5, 6)
>>> pdb.authors()
['N.WU', 'E.F.PAI']
```

molecuPy is a reasonably forgiving parser. If records are missing from the PDB file - even records which the PDB specification insists *must* be present - the file will still parse, and any missing properties will just be set to `None` or an empty list, whichever is appropriate.

2.2.3 Pdb Models

The heart of a Pdb is its model. A *Model* represents the structure contained in that PDB file, and is the environment in which all other molecules and structures are based.

All Pdb objects have a list of models, which in most cases will contain a single model. Structures created from NMR will often have multiple models - each containing the same structures but with slightly different coordinates. For ease of use, all Pdb objects also have a `model` method, which points to the first model in the list.

```
>>> pdb.models()
[<Model (3431 atoms)>]
>>> pdb.model()
<Model (3431 atoms)>
```

The Model class is an atomic structure (i.e. it inherits from *AtomicStructure*) which means you can get certain atomic properties directly from the model, such as mass, formula, and the atoms themselves:

```
>>> pdb.model().mass()
20630.8656
>>> pdb.model().formula()
Counter({'C': 2039, 'O': 803, 'N': 565, 'S': 22, 'P': 2})
>>> len(pdb.model().atoms())
3431
>>> pdb.model().get_atoms_by_element("P")
{<PdbAtom 3200 (P)>, <PdbAtom 3230 (P)>}
>>> pdb.get_atom_by_id(23)
<PdbAtom 23 (N)>
```

The complexes, chains and small molecules of the model exist as sets, and can be queried by ID or name:

```
>>> pdb.model().chains()
{<Chain B (214 residues)>, <Chain A (204 residues)>}
```

```
>>> len(pdb.model().small_molecules()) # Includes solvent molecules
184
>>> pdb.model().get_chain_by_id("B")
<Chain B (214 residues)>
>>> pdb.model().get_small_molecules_by_name("XMP")
{<SmallMolecule (XMP)>, <SmallMolecule (XMP)>}
```

Note: PDB files are not always perfect representations of the real molecular structures they are created from. Sometimes there are missing atoms, and sometimes there are missing residues. For this reason molecuPy draws a distinction between present and missing atoms, and present and missing residues. See the full API docs for more details.

2.2.4 Chains

A *Chain* object is an ordered sequence of Residue objects, and they are the macromolecular structures which constitute the bulk of the model.

```
>>> pdb.model().get_chain_by_id("A")
<Chain A (204 residues)>
>>> pdb.model().get_chain_by_id("A").chain_id()
'A'
>>> pdb.model().get_chain_by_id("A").residues()[0]
<Residue (VAL)>
```

Chains inherit from *ResiduicStructure* and *ResiduicSequence* and so have methods for retrieving residues:

```
>>> pdb.model().get_chain_by_id("A").get_residue_by_id("A23")
<Residue (ASN)>
>>> pdb.model().get_chain_by_id("A").get_residue_by_name("ASP")
<Residue (ASP)>
>>> pdb.model().get_chain_by_id("A").get_residues_by_name("ASN")
{<Residue A5 (ASN)>, <Residue A23 (ASN)>, <Residue A23A (ASN)>, <Residue A10
1(ASN)>, <Residue A141 (ASN)>, <Residue A199 (ASN)>}
>>> pdb.model().get_chain_by_id("A").sequence_string()
'VMNRLILAMDLMNRDDALRVTGEVREYIDTVKIGYPLVLSEGMDIIAEFRKRFGCRIIADFKVADIPETNEKICR
ATFKAGADAIIVHGFPGADSVRACLNVAEEMGREVFLLTEMSHPGAEMFIQGAADEIARMGVDLGVKNYVGPSTRP
ERLSRLREIIGQDSFLISPGGETLRFADAIIVGRSIYLADNPAAAAAGIIESI'
```

Like pretty much everything else in molecuPy, chains are ultimately atomic structures, and have the usual atomic structure methods for getting mass, retrieving atoms etc.

The *Residue* objects themselves are also atomic structures, and behave very similar to small molecules. They have `downstream_residue` and `upstream_residue` methods for getting the next and previous residue in their chain respectively.

2.2.5 Small Molecules

Many PDB files also contain non-macromolecular objects, such as ligands, and solvent molecules. In molecuPy, these are represented as *SmallMolecule* objects.

There's not a great deal to be said about small molecules. They are atomic structures, so you can get their mass, get atoms by name/ID etc.

```
>>> pdb.model().get_small_molecule_by_name("BU2")
<SmallMolecule A500 (BU2)>
>>> pdb.model().get_small_molecule_by_name("XMP").atoms()
{<PdbAtom 3240 (C)>, <PdbAtom 3241 (N)>, <PdbAtom 3242 (N)>, <PdbAtom 3243 (
C)>, <PdbAtom 3244 (O)>, <PdbAtom 3245 (C)>, <PdbAtom 3246 (O)>, <PdbAtom 32
47 (C)>, <PdbAtom 3248 (N)>, <PdbAtom 3249 (C)>, <PdbAtom 3250 (C)>, <PdbAto
m 3251 (O)>, <PdbAtom 3252 (C)>, <Atom 3253 (O)>, <PdbAtom 3230 (P)>, <PdbAt
om 3231 (O)>, <PdbAtom 3232 (O)>, <PdbAtom 3233 (O)>, <PdbAtom 3234 (O)>, <P
dbAtom 3235 (C)>, <PdbAtom 3236 (C)>, <PdbAtom 3237 (O)>, <Atom 3238 (C)>, <
PdbAtom 3239 (N)>}
>>> pdb.model().get_small_molecule_by_name("XMP").get_atom_by_id(3252)
<PdbAtom 3252 (C)>
```

The *BindSite* binding site of the molecule, if there is one, can be determined in one of two ways. If the PDB file already defines the site, it can be found with:

```
>>> pdb.model().get_small_molecule_by_name("XMP").bind_site()
<BindSite AC3 (11 residues)>
```

If there isn't one defined, you can try to predict it using atomic distances:

```
>>> pdb.model().get_small_molecule_by_name("XMP").predict_bind_site()
<BindSite CALC (5 residues)>
```

All atomic structures can do this, but it is perhaps most useful with small molecules.

2.2.6 Atoms

PDB structures - like everything else in the universe really - are ultimately collections of Atom - *Atom* - objects. They possess a few key properties from which much of everything else is created:

```
>>> pdb.model().get_atom_by_id(28)
<PdbAtom 28 (C)>
>>> pdb.model().get_atom_by_id(28).atom_id()
28
>>> pdb.model().get_atom_by_id(28).atom_name()
'CB'
>>> pdb.model().get_atom_by_id(28).element()
'C'
>>> pdb.model().get_atom_by_id(28).mass()
12.0107
```

molecuPy draws a distinction between generic atom objects, and `PdbAtom` objects, which have coordinates. These are the atoms listed in the PDB file as being observed in the experiment that produced it.

Why the distinction? PDB files also list *missing* atoms - atoms known to be present in the structure depicted but which were not observed in the data. For those the generic *Atom* class is used.

There are also missing residues, which are represented here as ordinary residues composed entirely of missing atoms. All residues have a `is_missing` method to make this clear.

The distance between any two PDB atoms can be calculated easily:

```
>>> atom1 = pdb.model().get_atom_by_id(23)
>>> atom2 = pdb.model().get_atom_by_id(28)
>>> atom1.distance_to(atom2)
7.931296047935668
```

Bonds will be assigned where possible - the bonds between atoms in standard residues are inferred from atom names, and PDB files contain annotations for other covalent bonds. These are assigned to the atoms as *Bond* objects.

```
>>> pdb.model().get_atom_by_id(27).bonds()
{<Bond between Atom 27 and Atom 101>, <Bond between Atom 100 and Atom 27>}
```

The atoms directly bonded to any atom can be obtained with `bonded_atoms`, and the set of all atoms that are *accessible* is accessed with `accessible_atoms`.

```
>>> pdb.model().get_atom_by_id(3201)
<PdbAtom 3200 (P)>
>>> pdb.model().get_atom_by_id(3201).bonded_atoms()
{<PdbAtom 3200 (P)>}
>>> pdb.model().get_atom_by_id(3200).bonded_atoms()
{<PdbAtom 3203 (O)>, <PdbAtom 3201 (O)>, <PdbAtom 3204 (O)>, <PdbAtom 3202 (
O)>}
>>> pdb.model().get_atom_by_id(3200).accessible_atoms()
{<PdbAtom 3214 (O)>, <PdbAtom 3215 (C)>, <PdbAtom 3216 (O)>, <PdbAtom 3217 (
C)>, <PdbAtom 3218 (N)>, <PdbAtom 3219 (C)>, <PdbAtom 3201 (O)>, <PdbAtom 32
20 (C)>, <PdbAtom 3202 (O)>, <PdbAtom 3221 (O)>, <PdbAtom 3203 (O)>, <PdbAto
m 3222 (C)>, <PdbAtom 3204 (O)>, <PdbAtom 3223 (O)>, <PdbAtom 3205 (C)>, <Pd
bAtom 3206 (C)>, <PdbAtom 3207 (O)>, <PdbAtom 3208 (C)>, <PdbAtom 3209 (N)>,
 <PdbAtom 3210 (C)>, <PdbAtom 3211 (N)>, <PdbAtom 3212 (N)>, <PdbAtom 3213 (
C)>}
```

Similarly, all atoms have a `model` method which refers back to their Model, and as long as this is the case, they can use their `local_atoms` method to return a set of all atoms within a given distance.

```
>>> pdb.model().get_atom_by_id(3201).local_atoms(5) # Atoms within 5A
{<PdbAtom 3214 (O)>, <PdbAtom 3215 (C)>, <PdbAtom 3216 (O)>, <PdbAtom 3217 (
C)>, <PdbAtom 3218 (N)>}
```

2.2.7 Binding Sites

BindSite objects represent binding sites. They are residuic structures, with the usual residuic structure methods, as well as a `ligand` property.

```
>>> pdb.model().sites()
{<BindSite AC2 (5 residues)>, <BindSite AC1 (4 residues)>, <BindSite AC4 (11
 residues)>, <BindSite AC3 (11 residues)>}
>>> pdb.model().get_site_by_id("AC1").residues()
{<Residue A10 (ASP)>, <Residue A11 (LEU)>, <Residue A34 (LYS)>}
>>> pdb.model().get_site_by_id("AC1").ligand()
<SmallMolecule A1000 (BU2)>
```

2.2.8 Secondary Structure

Chain objects have a `alpha_helices` property and a `beta_strands` property, which are sets of *AlphaHelix* objects and *BetaStrand* objects respectively.

2.2.9 Saving

Any model can be saved to file:

```
>>> model.save_as_pdb("filename.pdb")
```

2.3 Full API

2.3.1 molecupy.structures.atoms (Atoms)

This module contains classes for atoms and their bonds.

class molecupy.structures.atoms.**GhostAtom**(*element*, *atom_id*, *atom_name*)

This class represents atoms with no location. It is a 'ghost' in the sense that it is accounted for in terms of its mass, but it is 'not really there' because it has no location and cannot form bonds.

The reason for the distinction between ghost atoms and 'real' atoms comes from PDB files, where often not all the atoms in the studied molecule can be located in the (for example) electron density data and so there are no coordinates for them. They do 'exist' but they are missing from the PDB file coordinates.

They are described in terms of an Atom ID, an Atom name, and an element. They have mass but no location, and they can still be associated with molecules and models.

Parameters

- **element** (*str*) – The atom's element.
- **atom_id** (*int*) – The atom's id.
- **atom_name** (*str*) – The atom's name.

element(*element=None*)

Returns or sets the atom's element.

Parameters **element** (*str*) – If given, the atom's element will be set to this.

Return type str

atom_id()

Returns the atom's ID.

Return type int

atom_name(*atom_name=None*)

Returns or sets the atom's name.

Parameters **name** (*str*) – If given, the atom's name will be set to this.

Return type str

mass()

Returns the atom's mass

Return type float

molecule()

Returns the *SmallMolecule* or *Residue* the atom is a part of.

model()

Returns the *Model* the atom is a part of.

Return type Model

class molecupy.structures.atoms.**Atom**(*x*, *y*, *z*, **args*)

Base class: *GhostAtom*

Represents standard atoms which have Cartesian coordinates, and which can form bonds with other atoms.

They are distinguished from *GhostAtom* objects because they have a location in three dimensional space, though they inherit some properties from that more generic class of atom.

Parameters

- **x** (*float*) – The atom's x-coordinate.
- **y** (*float*) – The atom's y-coordinate.
- **z** (*float*) – The atom's z-coordinate.
- **element** (*str*) – The atom's element.
- **atom_id** (*int*) – The atom's id.
- **atom_name** (*str*) – The atom's name.

x (*x=None*)
Returns or sets the atom's x coordinate.

Parameters **x** (*float*) – If given, the atom's x coordinate will be set to this.

Return type float

y (*y=None*)
Returns or sets the atom's y coordinate.

Parameters **y** (*float*) – If given, the atom's y coordinate will be set to this.

Return type float

z (*z=None*)
Returns or sets the atom's z coordinate.

Parameters **z** (*float*) – If given, the atom's z coordinate will be set to this.

Return type float

location()
Returns the atom's xyz coordinates as a tuple.

Return type tuple

distance_to(*other_atom*)
Returns the distance between this atom and another, in Angstroms. Alternatively, an *AtomicStructure* can be provided and the method will return the distance between this atom and that structure's center of mass.

Parameters **other_atom** – The other atom or atomic structure.

Return type float

bonds()
The set of *Bond* objects belonging to this atom.

Returns set of *Bond* objects.

bond_to(*other_atom*)
Creates a *Bond* between this atom and another.

Parameters **other_atom** (Atom) – The other atom.

bonded_atoms()
The set of *Atom* objects bonded to this atom.

Returns set of *Atom* objects.

get_bond_with(*other_atom*)
Returns the specific *Bond* between this atom and some other atom, if it exists.

Parameters **other_atom** (Atom) – The other atom.

Return type Bond

break_bond_with(*other_atom*)
Removes the specific *Bond* between this atom and some other atom, if it exists.

Parameters **other_atom** (Atom) – The other atom.

accessible_atoms(*already_checked=None*)
The set of all *Atom* objects that can be accessed by following bonds.

Returns set of *Atom* objects.

local_atoms(*distance*, *include_hydrogens=True*)
Returns all *Atom* objects within a given distance of this atom (within a model).

Parameters

- **distance** – The cutoff in Angstroms to use.
- **include_hydrogens** (*bool*) – determines whether to include hydrogen atoms.

Returns set of *Atom* objects.

class molecupy.structures.atoms.**Bond**(*atom1*, *atom2*)
Represents a chemical bond between two *Atom* objects - covalent or ionic.

Parameters

- **atom1** (Atom) – The first atom.
- **atom2** (Atom) – The second atom.

atoms()
Returns the two atoms in this bond.

Returns set of *Atom*

bond_length()
The length of the bond in Angstroms.

Return type float

delete()
Removes the bond and updates the two atoms. Unless you have manually created some other reference, this will remove all references to the bond and it will eventually removed by garbage collection.

2.3.2 molecupy.structures.molecules (Atomic Structures)

Contains classes for simple structures made of atoms.

class molecupy.structures.molecules.**AtomicStructure**(**atoms*)
The base class for all structures which are composed of atoms.

Parameters **atoms** – A sequence of Atom objects.

atoms(*atom_type='localised'*)
Returns the atoms in this structure as a set.

Parameters **atom_type** (*str*) – The kind of atom to return. "all" will return all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.

Return type set

add_atom(*atom*)
Adds an atom to the structure.

Parameters **atom** (Atom) – The atom to add.

remove_atom(*atom*)
Removes an atom from the structure.

Parameters **atom** (Atom) – The atom to add.

mass(*atom_type='localised'*)
Returns the mass of the structure by summing the mass of all its atoms.

Parameters **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.

Return type float

formula(*atom_type='localised'*, *include_hydrogens=False*)
Retrurns the formula (count of each atom) of the structure.

Parameters

- **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type Counter

contacts_with(*other_atomic_structure*, *distance=4*, *include_hydrogens=True*)
Returns the set of all 'contacts' with another atomic structure, where a contact is defined as any atom-atom pair with an inter-atomic distance less than or equal to some number of Angstroms.

If the other atomic structure has atoms which are also in this atomic structure, those atoms will not be counted as part of the other structure.

Parameters

- **other_structure** (AtomicStructure) – The other atomic structure to compare to.
- **distance** – The distance to use (default is 4).
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type set of frozenset contacts.

internal_contacts(*distance=4*, *include_hydrogens=True*)
Returns the set of all atomic contacts within the atoms of an atomic structure, where a contact is defined as any atom-atom pair with an inter-atomic distance less than or equal to four Angstroms.

Contacts between atoms covalently bonded to each other will be ignored, as will contacts between atoms separated by just two covalent bonds.

Parameters

- **distance** – The distance to use (default is 4).
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type set of frozenset contacts.

predict_bind_site(*distance=5*, *include_hydrogens=True*)
Attempts to predict the residues that might make up the atomic structure's binding site by using atomic distances.

Parameters

- **distance** – The distance to use (default is 5s).
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type *BindSite* or None

translate(*x*, *y*, *z*)
Translates the structure in space.

Parameters

- **x** – The distance in Angstroms to move in the x-direction.
- **y** – The distance in Angstroms to move in the y-direction.
- **z** – The distance in Angstroms to move in the z-direction.

rotate(*axis*, *angle*)
Rotates the structure around an axis.

Parameters

- **axis** (*str*) – The axis to rotate around - must be "x", "y" or "z".
- **angle** – The angle, in degrees, to rotate by. Rotation is clockwise.

center_of_mass()
Returns the location of the structure's center of mass.

Return type tuple

radius_of_gyration()
The radius of gyration of an atomic structure is a measure of how extended it is. It is the root mean square deviation of the atoms from the structure's center of mass.

Return type float

get_atom_by_id(*atom_id*)
Retrurns the first atom that matches a given atom ID.

Parameters **atom_id** (*int*) – The atom ID to search by.

Return type *Atom* or None

get_atoms_by_element(*element*, *atom_type='localised'*)
Retruns all the atoms a given element.

Parameters

- **element** (*str*) – The element to search by.
- **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.

- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type set of *Atom* objects.

get_atom_by_element(*element*, *atom_type='localised'*)
Retrurns the first atom that matches a given element.

Parameters

- **element** (*str*) – The element to search by.
- **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type *Atom* or None

get_atoms_by_name(*atom_name*, *atom_type='localised'*)
Retruns all the atoms a given name.

Parameters

- **atom_name** (*str*) – The name to search by.
- **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type set of *Atom* objects.

get_atom_by_name(*atom_name*, *atom_type='localised'*)
Retrurns the first atom that matches a given name.

Parameters

- **atom_name** (*str*) – The name to search by.
- **atom_type** (*str*) – The kind of atom to use. "all" will use all atoms, "localised" just standard Atoms and "ghost" will just return generic *GhostAtom* atoms.
- **include_hydrogens** (*bool*) – determines whether hydrogen atoms should be included.

Return type *Atom* or None

class molecupy.structures.molecules.**SmallMolecule**(*molecule_id*, *molecule_name*, **atoms*)
Base class: *AtomicStructure*

Represents the ligands, solvent molecules, and other non-polymeric molecules in a structure.

Parameters

- **molecule_id** (*str*) – The molecule's ID.
- **molecule_name** (*str*) – The molecule's name.
- **atoms** – The molecule's atoms.

molecule_id()
Returns the molecule's ID.

Return type str

molecule_name(*molecule_name=None*)
Returns or sets the molecule's name.

Parameters **name** (*str*) – If given, the molecule's name will be set to this.

Return type str

bind_site(*bind_site=None*)
Returns or sets the molecule's *BindSite*.

Parameters **bind_site** (BindSite) – If given, the atom's bindsite will be set to this.

Return type BindSite

model()
Returns the *Model* that the molecule inhabits.

Return type Model

class molecupy.structures.molecules.**Residue**(*residue_id*, *residue_name*, **atoms*)
Base class: *AtomicStructure*

A Residue on a chain.

Parameters

- **residue_id** (*str*) – The residue's ID.
- **residue_name** (*str*) – The residue's name.
- **atoms** – The residue's atoms.

residue_id()
Returns the residue's ID.

Return type str

residue_name(*residue_name=None*)
Returns or sets the residue's name.

Parameters **name** (*str*) – If given, the residue's name will be set to this.

Return type str

chain()
Returns the *Chain* that the residue is within.

Return type Chain

is_missing()
Returns True if the residue was not observed in the experiment (and is therefore made up entirely of atoms with no coordinates).

Return type bool

downstream_residue()
Returns the residue connected to this residue's carboxy end.

Return type Residue

upstream_residue()
Returns the residue connected to this residue's amino end.

Return type Residue

connect_to(*downstream_residue*)
Connects this residue to a downstream residue.

Parameters **downstream_residue** (Residue) – The other residue.

disconnect_from(*other_residue*)
Breaks a connection with another residue.

Parameters **other_residue** (Residue) – The other residue.

alpha_carbon()
Attempts to retrieve the alpha carbon of the residue.

Return type Atom

2.3.3 molecupy.structures.chains (Residuic structures)

Contains classes for macrostructures made of residues.

class molecupy.structures.chains.**ResiduicStructure**(**residues*)
Base class: *AtomicStructure*

The base class for all structures which can be described as a set of residues.

Parameters **residues** – A sequence of *Residue* objects in this structure.

residues(*include_missing=True*)
Returns the residues in this structure as a set.

Parameters **include_missing** (*str*) – If False only residues present in the PDB coordinates will be returned, and not missing ones.

Return type set

add_residue(*residue*)
Adds a residue to the structure.

Parameters **residue** (Residue) – The residue to add.

remove_residue(*residue*)
Removes a residue from the structure.

Parameters **residue** (Residue) – The residue to remove.

get_residue_by_id(*residue_id*)
Returns the first residue that matches a given residue ID.

Parameters **residue_id** (*str*) – The residue ID to search by.

Return type *Residue* or None

get_residues_by_name(*residue_name*, *include_missing=True*)
Returns all the residues of a given name.

Parameters

- **residue_name** (*str*) – The name to search by.
- **include_missing** (*str*) – If False only residues present in the PDB coordinates will be returned, and not missing ones.

Return type set of *Residue* objects.

get_residue_by_name(*residue_name*, *include_missing=True*)
Returns the first residue that matches a given name.

Parameters

- **residue_name** (*str*) – The name to search by.

- **include_missing** (*str*) – If `False` only residues present in the PDB coordinates will be returned, and not missing ones.

Return type *Residue* or `None`

class `molecupy.structures.chains.`**ResiduicSequence**(**residues*)
Base class: *ResiduicStructure*

The base class for all structures which can be described as a sequence of residues.

Parameters **residues** – A sequence of *Residue* objects in this structure.

residues(*include_missing=True*)
Returns the residues in this structure as a `list`.

Parameters **include_missing** (*str*) – If `False` only residues present in the PDB coordinates will be returned, and not missing ones.

Return type `list`

add_residue(*residue*)
Adds a residue to the end of this sequence.

Parameters **residue** (Residue) – The residue to add.

sequence_string(*include_missing=True*)
Return the protein sequence of this chain as one letter codes.

Parameters **include_missing** (*str*) – If `False` only residues present in the PDB coordinates will be returned, and not missing ones.

Rtype str The protein sequence.

class `molecupy.structures.chains.`**Chain**(*chain_id*, **residues*)
Base class: *ResiduicSequence*

Represents chains - the polymeric units that make up most of PDB structures.

Parameters

- **chain_id** – The chain's ID.
- **residues** – The residues in this chain.

chain_id()
Returns the chain's ID.

Return type `str`

alpha_helices()
Returns the *AlphaHelix* objects on this chain.

Returns `set` of `AlphaHelix` objects

beta_strands()
Returns the `BetsStrand` objects on this chain.

Returns `set` of `BetaStrand` objects

model()
Returns the *Model* that the chain inhabits.

Return type `Model`

complex()
Returns the *Complex* that the chain is a part of.

Return type Model

get_helix_by_id(*helix_id*)
Returns the first alpha helix that matches a given helix ID.

Parameters **helix_id** (*str*) – The helix ID to search by.

Return type *AlphaHelix* or None

get_strand_by_id(*strand_id*)
Returns the first beta strand that matches a given strand ID.

Parameters **strand_id** (*str*) – The strand ID to search by.

Return type BetsStrand or None

class molecupy.structures.chains.**BindSite**(*site_id*, **residues*)
Base class: *ResiduicStructure*

Represents binding sites - the residue clusters that mediate ligand binding.

Parameters

- **site_id** – The site's ID.
- **residues** – The residues in this chain.

site_id()
Returns the site's ID.

Return type str

ligand(*ligand=None*)
Returns or sets the site's *SmallMolecule* ligand.

Parameters **ligand** (SmallMolecule) – If given, the ligand will be set to this.

Return type SmallMolecule

model()
Returns the *Model* that the site inhabits.

Return type Model

continuous_sequence()
If the residues are on the same chain, this will return a continuous sequence that contains all residues in this site, otherwise None.

Return type *ResiduicSequence*

class molecupy.structures.chains.**AlphaHelix**(*helix_id*, **residues*, *helix_class=None*, *comment=None*)
Base class: *ResiduicSequence*

Represents alpha helices.

Parameters

- **helix_id** (*str*) – The helix's ID.
- **residues** – The residues in this helix.
- **helix_class** (*str*) – The classification of the helix.
- **comment** (*str*) – Any comment associated with this helix.

helix_id()
Returns the helix's ID.

Return type str

helix_class(*helix_class=None*)
Returns or sets the helix's classification.

Parameters **helix_class** (*str*) – If given, the class will be set to this.

Return type str

comment(*comment=None*)
Returns or sets the helix's comment.

Parameters **comment** (*str*) – If given, the comment will be set to this.

Return type str

chain()
Returns the chain that this helix is on.

Return type Chain

class molecupy.structures.chains.**BetaStrand**(*strand_id*, *sense*, **residues*)
Base class: *ResiduicSequence*

Represents beta strands.

Parameters

- **strand_id** (*str*) – The strand's ID.
- **residues** – The residues in this strand.
- **sense** (*int*) – The sense of the strand with respect to the prior strand.

strand_id()
Returns the strand's ID.

Return type str

sense(*sense=None*)
Returns or sets the strand's sense with respect to the previous strand.

Parameters **sense** (*int*) – If given, the sense will be set to this.

Return type int

chain()
Returns the chain that this strand is on.

Return type Chain

2.3.4 molecupy.structures.complexes (Complexes)

Contains classes pertaining to complexes and multi-chain assemblies.

class molecupy.structures.complexes.**Complex**(*complex_id*, *complex_name*, **chains*)
Base class: *ResiduicStructure*

Represents complexes of multiple *Chain* objects.

Parameters

- **complex_id** (*str*) – The complex's unique ID.
- **complex_name** (*str*) – The complex's name.
- ***chains** – The chains to create the complex from.

complex_id()
Returns the complex's ID.

Return type str

complex_name(*complex_name=None*)
Returns or sets the complex's name.

Parameters **complex_name** (*str*) – If given, the complex's name will be set to this.

Return type str

chains()
Returns the *Chain* objects in this complex.

Returns set of *Chain* objects

model()
Returns the *Model* that the complex inhabits.

Return type Model

add_chain(*chain*)
Adds a *Chain* to the structure.

Parameters **chain** (Chain) – The chain to add.

remove_chain(*chain*)
Adds a *Chain* to the structure.

Parameters **chain** (Chain) – The chain to add.

2.3.5 molecupy.structures.models (The Model class)

Contains the Model class.

class molecupy.structures.models.**Model**
Base class: *AtomicStructure*

Represents the structural environment in which the other structures exist.

source()
The object the Model was created from.

small_molecules()
Returns all the *SmallMolecule* objects in this model.

Return type set

add_small_molecule(*small_molecule*)
Adds a small molecule to the model.

Parameters **small_molecule** (SmallMolecule) – The small molecule to add.

remove_small_molecule(*small_molecule*)
Removes a small molecule from the structure.

Parameters **small_molecule** (SmallMolecule) – The small molecule to remove.

get_small_molecule_by_id(*molecule_id*)
Returns the first small molecule that matches a given molecule ID.

Parameters **molecule_id** (*str*) – The molecule ID to search by.

Return type *SmallMolecule* or None

get_small_molecule_by_name(*molecule_name*)
Returns the first small molecules that matches a given name.

Parameters **molecule_name** (*str*) – The name to search by.

Return type *SmallMolecule* or None

get_small_molecules_by_name(*molecule_name*)
Returns all the small molecules of a given name.

Parameters **molecule_name** (*str*) – The name to search by.

Return type set of *SmallMolecule* objects.

duplicate_small_molecule(*small_molecule*, *molecule_id=None*)
Creates a copy of a small molecule in the Model. The coordinates will be identical but it will have a unique ID.

Parameters

- **small_molecule** (*SmalllMolecule*) – The molecule to duplicate.
- **molecule_id** (*str*) – If given, this will determine the ID of the new molecule.

chains()
Returns all the *Chain* objects in this model.

Return type set

add_chain(*chain*)
Adds a chain to the model.

Parameters **chain** (Chain) – The chain to add.

remove_chain(*chain*)
Removes a chain from the structure.

Parameters **chain** (Chain) – The chain to remove.

get_chain_by_id(*chain_id*)
Returns the first chain that matches a given chain ID.

Parameters **chain_id** (*str*) – The chain ID to search by.

Return type *Chain* or None

duplicate_chain(*chain*, *chain_id=None*)
Creates a copy of a chain in the Model. The coordinates will be identical but it will have a unique ID.

Parameters

- **chain** (Chain) – The chain to duplicate.
- **chain_id** (*str*) – If given, this will determine the ID of the new chain.

bind_sites()
Returns all the *BindSite* objects in this model.

Return type set

add_bind_site(*site*)
Adds a bind site to the model.

Parameters **site** (BindSite) – The bind site to add.

remove_bind_site(*site*)
Removes a bind site from the structure.

Parameters **site** (BindSite) – The bind site to remove.

get_bind_site_by_id(*site_id*)
Returns the first bind site that matches a given site ID.

Parameters **site_id** (*str*) – The site ID to search by.

Return type *BindSite* or None

complexes()
Returns all the *Complex* objects in this model.

Return type set

add_complex(*complex_*)
Adds a complex to the model.

Parameters **complex** (Complex) – The complex to add.

remove_complex(*complex_*)
Removes a complex from the model.

Parameters **complex** (Complex) – The complex to remove.

get_complex_by_id(*complex_id*)
Returns the first complex that matches a given complex ID.

Parameters **complex_id** (*str*) – The complex ID to search by.

Return type *Complex* or None

get_complex_by_name(*complex_name*)
Returns the first complex that matches a given name.

Parameters **complex_name** (*str*) – The name to search by.

Return type *Complex* or None

get_complexes_by_name(*complex_name*)
Returns all the complexes of a given name.

Parameters **complex_name** (*str*) – The name to search by.

Return type set of *Complex* objects.

duplicate_complex(*complex_*, *complex_id=None*, *complex_name=None*)
Creates a copy of a complex in the Model. The coordinates will be identical but it will have a unique ID.

Parameters

- **complex** (Complex) – The complex to duplicate.
- **complex_id** (*str*) – If given, this will determine the ID of the new complex.
- **complex_name** (*str*) – If given, this will determine the name of the new complex.

to_pdb_data_file()
Converts the Model to a *PdbDataFile*.

save_as_pdb(*path*)
Saves the Model to file as a PDB file.

Parameters **path** (*str*) – The location and file name to save as.

2.3.6 molecupy.pdb.pdbfile (PDB File)

This module is used to provide a container to the PDB file itself and its records - but not the data contained within them.

class molecupy.pdb.pdbfile.**PdbRecord**(*text*, *pdb_file=None*)

Represents the lines, or 'records' in a PDB file.

Indexing a PdbRecord will get the equivalent slice of the record text, only stripped, and converted to int or float if possible. Empty sub-strings will return None.

Parameters

- **text** (*str*) – The raw text of the record.
- **pdb_file** (PdbFile) – Optional: a *PdbFile* that the record should be associated with.

get_as_string(*start*, *end*)

Indexing a record will automatically convert the value to an integer or float if it can - using this method instead will force it to return a string.

Parameters

- **start** (*int*) – The start of the subsection.
- **end** (*int*) – The end of the subsection.

Return type str

number()

The record's line number in its associated *PdbFile*. If there is no file associated, this will return None.

Return type int

name(*name=None*)

The record's name (the first six characters). If a string value is supplied, the name will be set to the new value, and the text will also be updated.

Parameters **name** (*str*) – (optional) A new name to change to.

Return type str

content(*content=None*)

The record's text exlcuding the first six characters. If a string value is supplied, the content will be set to the new value, and the text will also be updated.

Parameters **content** (*str*) – (optional) A new content to change to.

Return type str

text(*text=None*)

The record's text, extended to 80 characters. If a string value is supplied, the text will be set to the new value, and the name and content will also be updated.

Parameters **text** (*str*) – (optional) A new text to change to.

Return type str

pdb_file(*pdb_file=None*)

The *PdbFile* that the record is associated with. This method can update the associated file by passing a *PdbFile* to it.

Parameters **pdb_file** (PdbFile) – (optional) A new *PdbFile* to set.

Return type PdbFile

class molecupy.pdb.pdbfile.**PdbFile**(*file_string=''*)
A PDB File - a representation of the file itself, with no processing of the data it contains (other than reading record names from the start of each line).

Parameters **file_string** (*str*) – The raw text of a PDB file.

source()
The object from which this PdbFile was created.

records()
A list of *PdbRecord* objects.

Returns list of *PdbRecord* objects.

get_record_by_name(*record_name*)
Gets the first *PdbRecord* of a given name.

Parameters **record_name** (*str*) – record name to search by.

Return type *PdbRecord* or None if there is no match.

get_records_by_name(*record_name*)
Gets all *PdbRecord* objects of a given name.

Parameters **record_name** (*str*) – record name to search by.

Returns list of *PdbRecord* objects.

add_record(*record*)
Adds a *PdbRecord* to the end of the list of records.

Parameters **record** (PdbRecord) – The *PdbRecord* to add.

remove_record(*record*)
Removes a *PdbRecord* from the list of records.

Parameters **record** (PdbRecord) – The *PdbRecord* to remove.

convert_to_string()
Converts the PdbFile to a string, that can be written to file.

to_pdb_data_file()
Converts the PdbFile to a *PdbDataFile*.

2.3.7 molecupy.pdb.pdbdatafile (PDB Data File)

This module performs the actual parsing of the PDB file, though it does not process the values that it extracts.

class molecupy.pdb.pdbdatafile.**PdbDataFile**
This object is essentially a list of values extracted from a PDB file. It functions as a data sheet.

source()
The object from which this PdbDataFile was created.

to_pdb_file()
Converts the PdbDataFile to a *PdbFile*.

classification(*classification=None*)
The classification of the PDB.

Parameters **classification** (*str*) – if given, the classifcation will be set to this.

Return type str

2.3.8 molecupy.pdb.pdb (PDBs)

This module contains creates the final Pdb object itself, and processes the data contained in the data file.

class molecupy.pdb.pdb.**Pdb**(*data_file*)

A representation of a PDB file and its contents, including the structure.

Parameters **data_file** (PdbDataFile) – The PDB data file with the parsed values.

data_file()

The *PdbDataFile* from which the object was created.

Return type PdbDataFile

classification()

The PDB classification.

Return type str

deposition_date()

The date the PDB was deposited.

Return type datetime.Date

pdb_code()

The PDB four-letter code.

Return type str

is_obsolete()

True if the PDB has been made obsolete by a newer PDB.

Return type bool

obsolete_date()

The date the PDB was made obsolete.

Return type datetime.Date

replacement_code()

The PDB code of the replacing PDB.

Return type str

title()

The title of the PDB.

Return type str

split_codes()

The PDB codes which complete this structure.

Return type list

caveat()

Any caveats for this structure.

Return type str

keywords()

Keywords for this PDB.

Return type list

experimental_techniques()

The experimental techniques used to produce this PDB.

Return type list

model_count()
The number of models in this PDB.

Return type int

model_annotations()
Annotations for the PDB's models.

Return type list

authors()
The PDB's authors.

Return type list

revisions()
Any changes made to the PDB file.

Return type list

supercedes()
The PDB codes that this PDB replaces.

Return type list

supercede_date()
The date this PDB replaced another.

Return type datetime.Date

journal()
The publication information for this PDB.

Return type dict

models()
The PDB's models.

Return type list

model()
The first *Model* in the PDB models.

Return type Model

2.3.9 molecupy.pdb.access (PDB Access)

This module contains the functions used to access PDB files themselves. These are the only functions to be imported into the top level directory, and so are all accesisble by importing molecupy itself.

molecupy.pdb.access.**pdb_from_string**(*text*)
Creates a *Pdb* object from the text of a PDB file.

Parameters **string** (*str*) – The raw text of a PDB file.

Return type *Pdb*

molecupy.pdb.access.**pdb_data_file_from_string**(*text*)
Creates a *PdbDataFile* object from the text of a PDB file.

Parameters **string** (*str*) – The raw text of a PDB file.

Return type *PdbDataFile*

molecupy.pdb.access.**pdb_file_from_string**(*text*)
Creates a *PdbFile* object from the text of a PDB file.

Parameters **string** (*str*) – The raw text of a PDB file.

Return type *PdbFile*

molecupy.pdb.access.**get_pdb_from_file**(*path*, *processing='pdb'*)
Creates a *Pdb*, *PdbDataFile*, or *PdbFile* from a file path on disk - the default behaviour being to create a *Pdb*.

Parameters

- **path** (*str*) – The location of the PDB file on disk.
- **processing** (*str*) – The level of processing you want the returned object to have. Propviding "pdbfile" will just return a *PdbFile*, "datafile" will return a *PdbDataFile*, and "pdb" (the default) will return a fully processed *Pdb* object.

Raises **FileNotFoundError** – if there is no file at the specified location.

molecupy.pdb.access.**get_pdb_remotely**(*code*, *processing='pdb'*)
Creates a *Pdb*, *PdbDataFile*, or *PdbFile* from a 4-letter PDB code - the default behaviour being to create a *Pdb*.

Parameters

- **code** (*str*) – The 4-letter PDB code.
- **processing** (*str*) – The level of processing you want the returned object to have. Propviding "pdbfile" will just return a *PdbFile*, "datafile" will return a *PdbDataFile*, and "pdb" (the default) will return a fully processed *Pdb* object.

Raises **InvalidPdbCodeError** – if there is no PDB with the given code.

2.3.10 molecupy.converters.pdbfile2pdbdatafile (PDB File to PDB Data File)

This module handles the logic of converting a *PdbFile* to a *PdbDataFile*

molecupy.converters.pdbfile2pdbdatafile.**pdb_data_file_from_pdb_file**(*pdb_file*)
Takes a *PdbFile*, converts it to a *PdbDataFile*, and returns it.

Parameters **pdb_file** (PdbFile) – The *PdbFile* to convert.

Return type *PdbDataFile*

molecupy.converters.pdbfile2pdbdatafile.**process_header_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the HEADER records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_obslte_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the OBSLTE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.

- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_title_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the TITLE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_split_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SPLIT records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_caveat_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the CAVEAT records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_compnd_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the COMPND records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_source_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SOURCE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_keywd_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the KEYWD records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_expdta_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the EXPDTA records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.

- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_nummdl_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the NUMMDL records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_mdltyp_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the MDLTYP records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_author_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the AUTHOR records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_revdat_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the REVDAT records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_sprsde_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SPRSDE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_jrnl_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the JRNL records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_remark_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the REMARK records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.

- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_dbref_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the DBREF records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_seqadv_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SEQADV records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_seqres_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SEQRES records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_modres_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the MODRES records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_het_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the HET records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_hetnam_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the HETNAM records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_hetsyn_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the HETSYN records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.

- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_formul_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the FORMUL records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_helix_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the HELIX records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_sheet_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SHEET records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_ssbond_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SSBOND records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_link_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the LINK records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_cispep_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the CISPEP records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_site_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SITE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_cryst1_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the CRYST1 records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_origx_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the ORIGX records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_scale_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the SCALE records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_mtrix_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the MTRIX records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_model_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the MODEL records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_atom_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the ATOM records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_anisou_records**(*data_file*, *pdb_file*)

Takes a *PdbDataFile* and updates it based on the ANISOU records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_ter_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the TER records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_hetatm_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the HETATM records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_conect_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the CONECT records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

molecupy.converters.pdbfile2pdbdatafile.**process_master_records**(*data_file*, *pdb_file*)
Takes a *PdbDataFile* and updates it based on the MASTER records in the provided *PdbFile*

Parameters

- **data_file** (PdbDataFile) – the Data File to update.
- **pdb_file** (PdbFile) – The source Pdb File

2.3.11 molecupy.converters.pdbdatafile2pdbfile (PDB Data File to PDB File)

This module handles the logic of converting a *PdbDataFile* to a *PdbFile*

molecupy.converters.pdbdatafile2pdbfile.**pdb_file_from_pdb_data_file**(*data_file*)
Takes a *PdbDataFile*, converts it to a *PdbFile*, and returns it.

Parameters **data_file** (PdbDataFile) – The *PdbDataFile* to convert.

Return type *PdbFile*

molecupy.converters.pdbdatafile2pdbfile.**create_compnd_records**(*pdb_file*, *data_file*)
Takes a *PdbFile* and creates COMPND records in it based on the data in the provided *PdbDataFile*

Parameters

- **pdb_file** (PdbFile) – the PDB File to update.
- **data_file** (PdbDataFile) – The source Pdb Data File

molecupy.converters.pdbdatafile2pdbfile.**create_atom_records**(*pdb_file*, *data_file*, *hetero=False*)
Takes a *PdbFile* and creates ATOM and HETATM records in it based on the data in the provided *PdbDataFile*

Parameters

- **pdb_file** (PdbFile) – the PDB File to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **hetero** (*bool*) – if True, the function will create HETATM records, and if False, ATOM records will be created. Default is False.

molecupy.converters.pdbdatafile2pdbfile.**create_conect_records**(*pdb_file*, *data_file*)

Takes a *PdbFile* and creates CONECT records in it based on the data in the provided *PdbDataFile*

Parameters

- **pdb_file** (PdbFile) – the PDB File to update.
- **data_file** (PdbDataFile) – The source Pdb Data File

2.3.12 molecupy.converters.pdbdatafile2model (PDB Data File to Model)

This module handles the logic of converting a *PdbDataFile* to a *Model*

molecupy.converters.pdbdatafile2model.**model_from_pdb_data_file**(*data_file*, *model_id=1*)

Takes a *PdbDataFile*, converts it to a *Model*, and returns it.

PdbDataFile objects can contain multiple models. By default, model 1 will be used, but you can specify specific models with the `model_id` argument.

Parameters

- **data_file** (PdbDataFile) – The *PdbDataFile* to convert.
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

Return type *Model*

molecupy.converters.pdbdatafile2model.**add_small_molecules_to_model**(*model*, *data_file*, *model_id*)

Takes a *Model* and creates *SmallMolecule* objects in it based on the `heteroatoms` in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**add_chains_to_model**(*model*, *data_file*, *model_id*)

Takes a *Model* and creates *Chain* objects in it based on the `atoms` in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**connect_atoms**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates *Bond* objects between atoms in it based on the connections in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**bond_residue_atoms**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates *Bond* objects within the residues of the Model, based on a pre-defined dictionary of how residues are connected internally.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**bond_residues_together**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates *Bond* objects between the residues of chains in the model.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**make_disulphide_bonds**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates disulphide *Bond* objects in it based on the ss_bonds in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**make_link_bonds**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates specified *Bond* objects in it based on the links in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**give_model_sites**(*model*, *data_file*, *model_id*)
Takes a *Model* and creates *BindSite* objects in it based on the sites in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File

- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**map_sites_to_ligands**(*model*, *data_file*, *model_id*)

Takes a *Model* and assocated ligands and binding sites to each other based on 800-remarks in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**give_model_alpha_helices**(*model*, *data_file*, *model_id*)

Takes a *Model* and creates *AlphaHelix* objects in it based on the helices in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**give_model_beta_strands**(*model*, *data_file*, *model_id*)

Takes a *Model* and creates *BetaStrand* objects in it based on the sheets in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

molecupy.converters.pdbdatafile2model.**give_model_complexes**(*model*, *data_file*, *model_id*)

Takes a *Model* and creates *Complex* objects in it based on the compounds in the provided *PdbDataFile*.

Parameters

- **model** (Model) – the model to update.
- **data_file** (PdbDataFile) – The source Pdb Data File
- **model_id** (*int*) – The ID of the model in the data fileto be used for conversion.

2.3.13 molecupy.converters.model2pdbdatafile (Model to PDB Data File)

This module handles the logic of converting a *Model* to a *PdbDataFile*

molecupy.converters.model2pdbdatafile.**pdb_data_file_from_model**(*model*)

Takes a *Model*, converts it to a PdbdataFile, and returns it.

Parameters **model** (Model) – The *Model* to convert.

Return type *PdbDataFile*

molecupy.converters.model2pdbdatafile.**add_complexes_to_data_file**(*data_file*, *model*)

Takes a *PdbDataFile* and updates its compounds based on the complexes in the provided *Model*

Parameters

- **data_file** (PdbDataFile) – the PDB Data File to update.
- **model** (PdbDataFile) – The source Model

molecupy.converters.model2pdbdatafile.**add_atoms_to_data_file**(*data_file*, *model*)
Takes a *PdbDataFile* and updates its atoms and heteroatoms based on the atoms in the provided *Model*

Parameters

- **data_file** (PdbDataFile) – the PDB Data File to update.
- **model** (PdbDataFile) – The source Model

molecupy.converters.model2pdbdatafile.**add_connections_to_data_file**(*data_file*, *model*)
Takes a *PdbDataFile* and updates its connections based on the bonds in the provided *Model*

Parameters

- **data_file** (PdbDataFile) – the PDB Data File to update.
- **model** (PdbDataFile) – The source Model

2.3.14 molecupy.exceptions (Exceptions)

molecuPy custom exceptions.

exception molecupy.exceptions.**LongBondWarning**
The warning issued if a covalent bond is made between two atoms that is unrealistically long.

exception molecupy.exceptions.**NoAtomsError**
The exception raised if an atomic structure is created without passing any atoms.

exception molecupy.exceptions.**NoResiduesError**
The exception raised if a residuic structure is created without passing any residues.

exception molecupy.exceptions.**MultipleResidueConnectionError**
The exception raised when a residue connection is made to a residue which is already connected to a residue in that fashion.

exception molecupy.exceptions.**BrokenHelixError**
The exception raised when an alpha helix is created with residues on different chains.

exception molecupy.exceptions.**BrokenStrandError**
The exception raised when a beta strand is created with residues on different chains.

exception molecupy.exceptions.**DuplicateAtomsError**
The exception raised if an atomic structure is created with two atoms of the same atom_id.

exception molecupy.exceptions.**DuplicateSmallMoleculesError**
The exception raised if a Model is given a small molecule when there is already a small molecule with that molecule_id.

exception molecupy.exceptions.**DuplicateResiduesError**
The exception raised if a residuic structure is created with two residues of the same residue_id.

exception molecupy.exceptions.**DuplicateChainsError**
The exception raised if a Model is given a chain when there is already a chain with that chain_id.

exception molecupy.exceptions.**DuplicateBindSitesError**
The exception raised if a Model is given a bindsite when there is already a site with that site_id.

exception molecupy.exceptions.**DuplicateComplexesError**
The exception raised if a Model is given a Complex when there is already a Complex with that complex_id.

exception molecupy.exceptions.**InvalidPdbCodeError**
The exception raised when a PDB file is requested that does not seem to exist.

2.4 Changelog

2.4.1 Release 1.1.0

29 January 2017

- Added PDB writing to file.
- Structures can now be translated and transformed.
- Complexes added.
- Models can now duplicate structures within them.
- Added center of mass and radius of gyration metrics.
- Atom distances can now be to a structure as well as another atom.
- Renamed different Atom types (there are now 'ghost atoms')

2.4.2 Release 1.0.3

15 August 2016

- Fixed bug relating to CONECT bonds sometimes bound to same atom.
- Fixed PDB datafile's string representation.

2.4.3 Release 1.0.2

12 August 2016

- Fixed bug relating to bind site construction from invalid chain.
- Fixed bug relating to disulphide bonds sometimes bound to same atom.

2.4.4 Release 1.0.1

4 August 2016

- Version number fix.

2.4.5 Release 1.0.0

4 August 2016

- A backwards-incompatible redesign of molecuPy.
- Attributes are now methods.
- Bind site calculation is now done at the atomic structure level.
- Tests are now fully mocked and easier to establish.

- Atoms can now detect nearby atoms as long as they are in the same model.

2.4.6 Release 0.4.1

11 July 2016

- Bug fix
  - Fixed bug where occasionally covalent bonds would be made over missing residues.

2.4.7 Release 0.4.0

20 June 2016

- Secondary Structure
  - Added Alpha Helix class.
  - Added Beta Strand class.
- Residue distance matrices
  - Chains can now generate SVG distance matrices showing the distances between residues.
- Missing residues
  - Chains can now produce a combined list of all residue IDs, missing and present.

2.4.8 Release 0.3.0

1 June 2016

- Atom connectivity
  - Covalent bonds are now added, and atoms now know about their neighbours.
- Residue connectivity
  - Residues are now aware of which residue they are covalently bound to in their chain.
- Atomic contacts
  - Added methods for calculating the internal and external atomic contacts of any atomic structure.
- Bug fixes
  - Fixed bug where PDB files could not have site mapping parsed where there was no space between the chain ID and residue ID.

2.4.9 Release 0.2.0

19 May 2016

- Protein Sequences
  - Residuic Sequences can now return their amino acid sequence as a string
- Binding Sites
  - Added a class for binding sites

  - Mapped sites to ligands
  - Added methods for getting sites for ligands
- Insert codes
  - Incorporated insert codes into residue IDs

2.4.10 Release 0.1.0

16 May 2016

- Basic PDB parsing
  - Models
  - Chains
  - Residues
  - Atoms
  - Small Molecules

m

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

X

Y

Z