CAT Documentation

Release 0.5.5

B. F. van Beek

Sep 24, 2019

CONTENTS

Contents:

CHAPTER

ONE

COMPOUND ATTACHMENT TOOL 0.5.5

CAT is a collection of tools designed for the construction of various chemical compounds. Further information is provided in the documentation.

1.1 Installation

- Download miniconda for python3: miniconda (also you can install the complete anaconda version).
- Install according to: installConda.
- Create a new virtual environment, for python 3.7, using the following commands:
  - `conda create --name CAT python`
- The virtual environment can be enabled and disabled by, respectively, typing:
  - Enable: `conda activate CAT`
  - Disable: `conda deactivate`

1.1.1 Dependencies installation

Using the conda environment the following packages should be installed:

- rdkit : `conda install -y --name CAT --channel conda-forge rdkit`

1.1.2 Package installation

Finally, install **CAT** using pip:

- **CAT**: `pip install git+https://github.com/nlesc-nano/CAT@master --upgrade`

Now you are ready to use **CAT**.

1.2 Input files

Running **CAT** and can be done with the following command: `init_cat my_settings.yaml`. The user merely has to provide a yaml file with the job settings, settings which can be tweaked and altered to suit ones purposes (see example1). Alternatively, **CAT** can be run like a regular python script, bypassing the command-line interface (*i.e.* `python input.py`, see example2).

An extensive description of the various available settings is available in the documentation.

CHAPTER

TWO

CAT DOCUMENTATION

For a more detailed description of the **CAT** compound builder read the documentation. The documentation is divided into three parts: The basics, further details about the input cores & ligands and finally a more detailed look into the customization of the various jobs.

2.1 General Overview & Getting Started

A basic recipe for running **CAT**:

1. Create two directories named 'core' and 'ligand'. The 'core' directory should contain the input cores & the 'ligand' should contain the input ligands. The quantum dots will be exported to the 'QD' directory.

2. Customize the job settings to your liking, see CAT/examples/input_settings.yaml for an example. Note: everything under the `optional` section does **not** have to be included in the input settings. As is implied by the name, everything in `optional` is completely optional.

3. Run **CAT** with the following command: `init_cat input_settings.yaml`

4. Congratulations, you just ran **CAT**!

The default **CAT** settings, at various levels of verbosity, are provided below.

2.1.1 Default Settings

```
path: None

input_cores:
    - Cd68Se55.xyz:
        guess_bonds: False

input_ligands:
    - OC(C)=O
    - OC(CC)=O
```

2.1.2 Verbose default Settings

```
path: None

input_cores:
    - Cd68Se55.xyz:
```

(continues on next page)

(continued from previous page)

```
        guess_bonds: False

input_ligands:
    - OC(C)=O
    - OC(CC)=O

optional:
    database:
        dirname: database
        read: True
        write: True
        overwrite: False
        mol_format: (pdb, xyz)
        mongodb: False

    core:
        dirname: core
        dummy: Cl

    ligand:
        dirname: ligand
        optimize: True
        split: True
        functional_groups: null
        cosmo-rs: False

    qd:
        dirname: QD
        optimize: False
        activation_strain: False
        dissociate: False
```

2.1.3 Maximum verbose default Settings

```
path: None

input_cores:
    - Cd68Se55.xyz:
        guess_bonds: False

input_ligands:
    - OC(C)=O
    - OC(CC)=O

optional:
    database:
        dirname: database
        read: (core, ligand, qd)
        write: (core, ligand, qd)
        overwrite: False
        mol_format: (pdb, xyz)
        mongodb: False

    core:
        dirname: core
```

(continues on next page)

(continued from previous page)

```
        dummy: Cl

    ligand:
        dirname: ligand
        optimize: True
        split: True
        functional_groups: null
        cosmo-rs: False

    qd:
        dirname: QD
        optimize: False
        activation_strain: False
        dissociate:
            core_atom: Cd
            lig_count: 2
            keep_files: True
            core_core_dist: 5.0
            lig_core_dist: 5.0
            topology: {}

            job1: False
            s1: False
            job2: False
            s2: False
```

2.2 path

2.2.1 Default Settings

```
path: null
```

2.2.2 Arguments

path

Parameter

- **Type** - `str` or `NoneType`
- **Default value** – `None`

The path were all working directories are/will be stored. To use the current working directory, use one of the following values: `None`, `"."`, `""` or `"path_to_workdir"`.

Note: The yaml format uses `null` rather than `None` as in Python.

2.3 input_cores & input_ligands

Thia section related relates the importing and processing of cores and ligands. Ligand & cores can be imported from a wide range of different files and files types, which can roughly be divided into three categories:

1. Files containing coordinates of a single molecule: .xyz, .pdb & .mol files.
2. Python objects: `plams.Molecule`, `rdkit.Chem.Mol` & SMILES strings (`str`).
3. Containers with one or multiple input molecules: directories & .txt files.

In the later case, the container can consist of multiple SMILES strings or paths to .xyz, .pdb and/or .mol files. If necessary, containers are searched recursively. Both absolute and relative paths are explored.

2.3.1 Default Settings

```
input_cores:
    - Cd68Se55.xyz:
        guess_bonds: False

input_ligands:
    - OC(C)=O
    - OC(CC)=O
    - OC(CCC)=O
    - OC(CCCC)=O
```

2.3.2 Optional arguments

.guess_bonds

Parameter

- **Type** - `bool`
- **Default value** – `False`

Try to guess bonds and bond orders in a molecule based on the types atoms and the relative of atoms. Is set to False by default, with the exception of .xyz files.

.column

Parameter

- **Type** - `int`
- **Default value** – `0`

The column containing the to be imported molecules. Relevant when importing structures from .txt and .xlsx files with multiple columns. Relevant for .txt and .csv files. Numbering starts from 0.

.row

Parameter

- **Type** - `int`
- **Default value** – `0`

The first row in a column which contains a molecule. Useful for when, for example, the very first row contains the title of aforementioned row, in which case row = 1 would be a sensible choice. Relevant for .txt and .csv files. Numbering starts from 0.

.indices

Parameter

- **Type** - `int` or `tuple` [`int`]
- **Default value** – `None`

The behaviour of this argument depends on whether it is passed to a molecule in *input_cores* or *input_ligands*:

input_cores
Manually specify the atomic index of one ore more atom(s) in the core that will be replaced with ligands. If left empty, all atoms of a user-specified element (see `optional.cores.dummy`) will be replaced with ligands.

input_ligands
Manually specify the atomic index of the ligand atom that will be attached to core (implying argument_dict: *optional.ligand.split* = `False`). If two atomic indices are provided (*e.g.* `(1, 2)`), the bond between atoms `1` and [`2`] will be broken and the remaining molecule containing atom `2` is attached to the core, (implying argument_dict: *split* = `True`). Serves as an alternative to the functional group based `CAT.find_substructure()` function, which identifies the to be attached atom based on connectivity patterns (*i.e.* functional groups).

Note: Atom numbering follows the PLAMS [1, 2] convention of starting from 1 rather than 0.

2.4 Optional

There are a number of arguments which can be used to modify the functionality and behaviour of the quantum dot builder. Herein an overview is provided.

Note: Inclusion of this section in the input file is not required, assuming one is content with the default settings.

2.4.1 Index

Option	Description
optional.database.dirname	The name of the directory where the database will be stored.
optional.database.read	Attempt to read results from the database before starting calculations.
optional.database.write	Export results to the database.
optional.database.overwrite	Allow previous results in the database to be overwritten.
optional.database.mol_format	The file format(s) for exporting moleculair structures.
optional.database.mongodb	Options related to the MongoDB format.
optional.core.dirname	The name of the directory where all cores will be stored.
optional.core.dummy	Atomic number of symbol of the core dummy atoms.
optional.ligand.dirname	The name of the directory where all ligands will be stored.
optional.ligand.optimize	Optimize the geometry of the to-be attached ligands.
optional.ligand.functional_groups	Manually specify SMILES strings representing functional groups.
optional.ligand.split	If the ligand should be attached in its entirety to the core or not.
optional.ligand.cosmo-rs	Perform a property calculation with COSMO-RS on the ligand.
optional.qd.dirname	The name of the directory where all quantum dots will be stored.
optional.qd.optimize	Optimize the quantum dot (i.e. core + all ligands) .
optional.qd.activation_strain	Perform an activation strain analyses.
optional.qd.dissociate	Calculate the ligand dissociation energy.

2.4.2 Default Settings

```
optional:
    database:
        dirname: database
        read: True
        write: True
        overwrite: False
        mol_format: (pdb, xyz)
        mongodb: False

    core:
        dirname: core
        dummy: Cl

    ligand:
        dirname: ligand
        optimize: True
        functional_groups: null
        split: True
        cosmo-rs: False

    qd:
        dirname: qd
        optimize: False
        activation_strain: False
        dissociate: False
```

2.4.3 Arguments

Database

optional.**database**
: All database-related settings.

Note: For *optional.database* settings to take effect the Data-CAT package has to be installed.

Example:

```
optional:
    database:
        dirname: database
        read: True
        write: True
        overwrite: False
        mol_format: (pdb, xyz)
        mongodb: False
```

optional.database.**dirname**

Parameter

- **Type** - `str`
- **Default Value** - `"database"`

The name of the directory where the database will be stored.

The database directory will be created (if it does not yet exist) at the path specified in *path*.

optional.database.**read**

Parameter

- **Type** - `bool`, `str` or `tuple` [`str`]
- **Default value** - `("core", "ligand", "qd")`

Attempt to read results from the database before starting calculations.

Before optimizing a structure, check if a geometry is available from previous calculations. If a match is found, use that structure and avoid a geometry reoptimizations. If one wants more control then the boolean can be substituted for a list of strings (*i.e.* `"core"`, `"ligand"` and/or `"qd"`), meaning that structures will be read only for a specific subset.

Example

Example #1:

```
optional:
    database:
        read: (core, ligand, qd)  # This is equivalent to read: True
```

Example #2:

```
optional:
    database:
        read: ligand
```

optional.database.**write**

Parameter

- **Type** - bool, str or tuple [str]
- **Default value** - ("core", "ligand", "qd")

Export results to the database.

Previous results will **not** be overwritten unless *optional.database.overwrite* = True. If one wants more control then the boolean can be substituted for a list of strings (*i.e.* "core", "ligand" and/or "qd"), meaning that structures written for for a specific subset.

See *optional.database.read* for a similar relevant example.

optional.database.**overwrite**

Parameter

- **Type** - bool, str or tuple [str]
- **Default value** - False

Allow previous results in the database to be overwritten.

Only apllicable if *optional.database.write* = True. If one wants more control then the boolean can be substituted for a list of strings (*i.e.* "core", "ligand" and/or "qd"), meaning that structures written for for a specific subset.

See *optional.database.read* for a similar relevant example.

optional.database.**mol_format**

Parameter

- **Type** - bool, str or tuple [str]
- **Default value** - ("pdb", "xyz")

The file format(s) for exporting moleculair structures.

By default all structures are stored in the .hdf5 format as (partially) de-serialized .pdb files. Additional formats can be requisted with this keyword. Accepted values: "pdb", "xyz", "mol" and/or "mol2".

optional.database.**mongodb**

Parameter

- **Type** - bool or dict
- **Default Value** – False

Options related to the MongoDB format.

See also

More extensive options for this argument are provided in *The Database Class*:.

Core

optional.**core**
All settings related to the core.

Example:

```
optional:
    core:
        dirname: core
        dummy: Cl
```

optional.core.**dirname**

Parameter

- **Type** - str
- **Default value** – "core"

The name of the directory where all cores will be stored.

The core directory will be created (if it does not yet exist) at the path specified in *path*.

optional.core.**dummy**

Parameter

- **Type** - str or int
- **Default value** – 17

Atomic number of symbol of the core dummy atoms.

The atomic number or atomic symbol of the atoms in the core which are to be replaced with ligands. Alternatively, dummy atoms can be manually specified with the core_indices variable.

Ligand

optional.**ligand**
All settings related to the ligands.

Example:

```
optional:
    ligand:
        dirname: ligand
        optimize: True
        functional_groups: null
```

(continues on next page)

(continued from previous page)

```
        split: True
        cosmo-rs: False
```

optional.ligand.**dirname**

Parameter

- **Type** - `str`
- **Default value** – `"ligand"`

The name of the directory where all ligands will be stored.

The ligand directory will be created (if it does not yet exist) at the path specified in *path*.

optional.ligand.**optimize**

Parameter

- **Type** - `bool`
- **Default value** – `True`

Optimize the geometry of the to-be attached ligands.

The ligand is split into one or multiple (more or less) linear fragments, which are subsequently optimized (RDKit UFF [1, 2, 3]) and reassembled while checking for the optimal dihedral angle. The ligand fragments are biased towards more linear conformations to minimize inter-ligand repulsion once the ligands are attached to the core.

optional.ligand.**functional_groups**

Parameter

- **Type** - `str` or `tuple` [`str`]
- **Default value** – `None`

Manually specify SMILES strings representing functional groups.

For example, with *optional.ligand.functional_groups* = `("O[H]", "[N+].[Cl-]")` all ligands will be searched for the presence of hydroxides and ammonium chlorides.

The first atom in each SMILES string (*i.e.* the "anchor") will be used for attaching the ligand to the core, while the last atom (assuming *optional.ligand.split* = `True`) will be dissociated from the ligand and disgarded.

If not specified, the default functional groups of **CAT** are used.

Note: This argument has no value be default and will thus default to SMILES strings of the default functional groups supported by **CAT**.

Note: The yaml format uses `null` rather than `None` as in Python.

---

optional.ligand.**split**

Parameter

- **Type** - `bool`
- **Default value** – `True`

If `False`: The ligand is to be attached to the core in its entirety .

Before	After
$NR_4{}^+$	$NR_4{}^+$
O_2CR	O_2CR
HO_2CR	HO_2CR
H_3CO_2CR	H_3CO_2CR

`True`: A proton, counterion or functional group is to be removed from the ligand before attachment to the core.

Before	After
$Cl^- + NR_4{}^+$	$NR_4{}^+$
HO_2CR	O_2CR^-
$Na^+ + O_2CR^-$	O_2CR^-
HO_2CR	O_2CR^-
H_3CO_2CR	O_2CR^-

optional.ligand.cosmo-rs

Parameter

- **Type** - `bool` or `dict`
- **Default value** – `False`

Perform a property calculation with COSMO-RS [4, 5, 6, 7] on the ligand.

The COSMO surfaces are by default constructed using ADF MOPAC [8, 9, 10].

The solvation energy of the ligand and its activity coefficient are calculated in the following solvents: acetone, acetonitrile, dimethyl formamide (DMF), dimethyl sulfoxide (DMSO), ethyl acetate, ethanol, *n*-hexane, toluene and water.

QD

optional.**qd**

All settings related to the quantum dots.

Example:

```
optional:
    qd:
        dirname: QD
        optimize: False
        activation_strain: False
        dissociate: False
```

`optional.qd.`**`dirname`**

Parameter

- **Type** - `str`
- **Default value** – `"qd"`

The name of the directory where all quantum dots will be stored.

The quantum dot directory will be created (if it does not yet exist) at the path specified in *path*.

`optional.qd.`**`optimize`**

Parameter

- **Type** - `bool` or `dict`
- **Default value** – `False`

Optimize the quantum dot (i.e. core + all ligands) .

By default the calculation is performed with ADF UFF [3, 11]. The geometry of the core and ligand atoms directly attached to the core are frozen during this optimization.

`optional.qd.`**`activation_strain`**

Parameter

- **Type** - `bool`
- **Default value** – `False`

Perform an activation strain analyses [12, 13, 14].

The activation strain analyses (kcal mol^{-1}) is performed on the ligands attached to the quantum dot surface with RDKit UFF [1, 2, 3].

The core is removed during this process; the analyses is thus exclusively focused on ligand deformation and inter-ligand interaction. Yields three terms:

1. dE_{strain} : The energy required to deform the ligand from their equilibrium geometry to the geometry they adopt on the quantum dot surface. This term is, by definition, destabilizing. Also known as the preperation energy (dE_{prep}).

2. dE_{int} : The mutual interaction between all deformed ligands. This term is characterized by the non-covalent interaction between ligands (UFF Lennard-Jones potential) and, depending on the inter-ligand distances, can be either stabilizing or destabilizing.

3. dE : The sum of dE_{strain} and dE_{int}. Accounts for both the destabilizing ligand deformation and (de-)stabilizing interaction between all ligands in the absence of the core.

`optional.qd.`**`dissociate`**

Parameter

- **Type** - `bool` or `dict`
- **Default value** – `False`

Calculate the ligand dissociation energy.

Calculate the ligand dissociation energy (BDE) of ligands attached to the surface of the core. See *Bond Dissociation Energy* for more details. The calculation consists of five distinct steps:

1. Dissociate all combinations of n ligands (Y) and an atom from the core (X) within a radius r from aforementioned core atom. The dissociated compound has the general structure of XY_n.

2. Optimize the geometry of XY_n at the first level of theory (1). Default: ADF MOPAC [1, 2, 3].

3. Calculate the "electronic" contribution to the BDE (ΔE) at the first level of theory (1): ADF MOPAC [1, 2, 3]. This step consists of single point calculations of the complete quantum dot, XY_n and all XY_n-dissociated quantum dots.

4. Calculate the thermalchemical contribution to the BDE ($\Delta\Delta G$) at the second level of theory (2). Default: ADF UFF [4, 5]. This step consists of geometry optimizations and frequency analyses of the same compounds used for step 3.

5. $\Delta G_{tot} = \Delta E_1 + \Delta\Delta G_2 = \Delta E_1 + (\Delta G_2 - \Delta E_2)$.

See also

More extensive options for this argument are provided in *Bond Dissociation Energy*:.

2.5 Bond Dissociation Energy

Calculate the bond dissociation energy (BDE) of ligands attached to the surface of the core. The calculation consists of five distinct steps:

1. Dissociate all combinations of n ligands (Y, see `optional.qd.dissociate.lig_count`) a nd an atom from the core (X, see `optional.qd.dissociate.core_atom`) within a radius r from aforementioned core atom (see `optional.qd.dissociate.lig_core_dist` and `optional.qd.dissociate.core_core_dist`). The dissociated compound has the general structure of XY_n.

2. Optimize the geometry of XY_n at the first level of theory (1). Default: ADF MOPAC [1, 2, 3].

3. Calculate the "electronic" contribution to the BDE (ΔE) at the first level of theory (1): ADF MOPAC [1, 2, 3]. This step consists of single point calculations of the complete quantum dot, XY_n and all XY_n-dissociated quantum dots.

4. Calculate the thermalchemical contribution to the BDE ($\Delta\Delta G$) at the second level of theory (2). Default: ADF UFF [4, 5]. This step consists of geometry optimizations and frequency analyses of the same compounds used for step 3.

5. $\Delta G_{tot} = \Delta E_1 + \Delta\Delta G_2 = \Delta E_1 + (\Delta G_2 - \Delta E_2)$.

2.5.1 Default Settings

```yaml
optional:
    qd:
        dissociate:
            core_atom: Cd
            lig_count: 2
            keep_files: True
            core_core_dist: 5.0  # Ångström
            lig_core_dist: 5.0  # Ångström
            core_index: False
            topology: {}
```

(continues on next page)

(continued from previous page)

```
        job1: AMSJob
        s1: True
        job2: AMSJob
        s2: True
```

2.5.2 Arguments

optional.qd.**dissociate**

```
optional:
    qd:
        dissociate:
            core_atom: Cd
            lig_count: 2
            keep_files: True
            core_core_dist: 5.0  # Ångström
            lig_core_dist: 5.0  # Ångström
            core_index: False
            topology:
                7: vertice
                8: edge
                10: face
```

optional.qd.dissociate.**core_atom**

Parameter

- **Type** - str or int
- **Default value** – None

The atomic number or atomic symbol of the core atoms (X) which are to be dissociated. The core atoms are dissociated in combination with n ligands (Y, see *optional.qd.dissociate.lig_count*). Yields a compound with the general formula XY_n.

If one is interested in dissociating ligands in combination with a molecular species (*e.g.* $X = NR_4{}^+$) the atomic number (or symbol) can be substituted for a SMILES string represting a polyatomic ion (*e.g.* tetramethyl ammonium: C[N+](C)(C)C).

If a SMILES string is provided it must satisfy the following 2 requirements:

1. The SMILES string *must* contain a single charged atom; unpredictable behaviour can occur otherwise.
2. The provided structure (including its bonds) must be present in the core.

Warning: This argument has no value be default and thus *must* be provided by the user.

Note: The yaml format uses `null` rather than `None` as in Python.

`optional.qd.dissociate.`**`lig_count`**

Parameter

- **Type** - `int`
- **Default value** – `None`

The number of ligands, n, which is to be dissociated in combination with a single core atom (X, see *optional.qd.dissociate.core_atom*). Yields a compound with the general formula XY_n.

Warning: This argument has no value be default and thus *must* be provided by the user.

Note: The yaml format uses `null` rather than `None` as in Python.

`optional.qd.dissociate.`**`keep_files`**

Parameter

- **Type** - `bool`
- **Default value** – `True`

Whether to keep or delete all BDE files after all calculations are finished.

`optional.qd.dissociate.`**`core_core_dist`**

Parameter

- **Type** - `float` or `int`
- **Default value** – `0.0`

The maximum to be considered distance (Ångström) between atoms in *optional.qd.dissociate.core_atom*. Used for determining the topology of the core atom (see *optional.qd.dissociate.topology*) and whether it is exposed to the surface of the core or not. It is recommended to use a radius which encapsulates a single (complete) shell of neighbours.

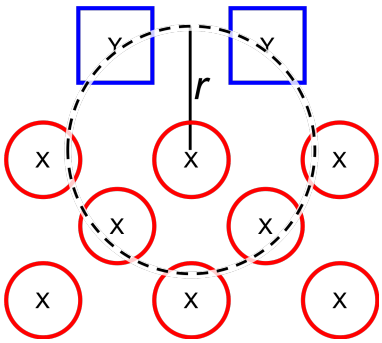
If not specified (or equal to `0.0`) **CAT** will attempt to guess a suitable value based on the cores' radial distribution function.

`optional.qd.dissociate.`**`lig_core_dist`**

Parameter

- **Type** - `float` or `int`
- **Default value** – `5.0`

Dissociate all possible combinations of n ligands and a single core atom (see *optional.qd.dissociate.core_atom*) within a given radius (Ångström) from aforementioned core atom. The number of ligands dissociated in combination with a single core atom is controlled by *optional.qd.dissociate.lig_count*.



optional.qd.dissociate.**core_index**

Parameter

- **Type** - int or tuple [int]
- **Default value** – None

Alternative to *optional.qd.dissociate.lig_core_dist* and *optional.qd.dissociate.core_atom*. Manually specify the indices of all to-be dissociated atoms in the core. Core atoms will be dissociated in combination with the n closest ligands.

Note: Atom numbering follows the PLAMS [1, 2] convention of starting from 1 rather than 0.

Note: The yaml format uses null rather than None as in Python.

---

optional.qd.dissociate.**topology**

Parameter

- **Type** - dict
- **Default value** – {}

A dictionary which translates the number neighbouring core atoms (see *optional.qd.dissociate.core_atom* and *optional.qd.dissociate.core_core_dist*) into a topology. Keys represent the number of neighbours, values represent the matching topology.

Example

Given a *optional.qd.dissociate.core_core_dist* of 5.0 Ångström, the following options can be interpreted as following:

```
optional:
    qd:
        dissociate:
            7: vertice
            8: edge
            10: face
```

Core atoms with `7` other neighbouring core atoms (within a radius of `5.0` Ångström) are marked as `"vertice"`, the ones with `8` neighbours are marked as `"edge"` and the ones with `10` neighbours as `"face"`.

2.5.3 Arguments - Job Customization

`optional.qd.`**`dissociate`**

```
optional:
    qd:
        dissociate:
            job1: AMSJob
            s1: True
            job2: AMSJob
            s2: True
```

`optional.qd.dissociate.`**`job1`**

Parameter

- **Type** - `type`, `str` or `bool`
- **Default value** – `plams.AMSJob`

A `type` object of a `plams.Job` subclass, used for calculating the "electronic" component (ΔE_1) of the bond dissociation energy. Involves single point calculations.

Alternatively, an alias can be provided for a specific job type (see *Type Aliases*).

Setting it to `True` will default to `plams.AMSJob`, while `False` is equivalent to *`optional.qd.dissociate`* = `False`.

`optional.qd.dissociate.`**`s1`**

Parameter

- **Type** - `dict`, `str` or `bool`
- **Default value** – See below

```
s1:
    input:
        mopac:
            model: PM7
        ams:
            system:
                charge: 0
```

The job settings used for calculating the "electronic" component (ΔE_1) of the bond dissociation energy.

Alternatively, a path can be provided to .json or .yaml file containing the job settings.

Setting it to `True` will default to the `["MOPAC"]` block in CAT/data/templates/qd.yaml, while `False` is equivalent to *optional.qd.dissociate* = `False`.

`optional.qd.dissociate.`**`job2`**

Parameter

- **Type** - `type`, `str` or `bool`
- **Default value** – `plams.AMSJob`

A `type` object of a `plams.Job` subclass, used for calculating the thermal component ($\Delta\Delta G_2$) of the bond dissociation energy. Involves a geometry reoptimizations and frequency analyses.

Alternatively, an alias can be provided for a specific job type (see *Type Aliases*).

Setting it to `True` will default to `plams.AMSJob`, while `False` will skip the thermochemical analysis completely.

`optional.qd.dissociate.`**`s1`**

Parameter

- **Type** - `dict`, `str` or `bool`
- **Default value** – See below

```
s2:
    input:
        uff:
            library: uff
        ams:
            system:
                charge: 0
                bondorders:
                    _1: null
```

The job settings used for calculating the thermal component ($\Delta\Delta G_2$) of the bond dissociation energy.

Alternatively, a path can be provided to .json or .yaml file containing the job settings.

Setting it to `True` will default to the the *MOPAC* block in CAT/data/templates/qd.yaml, while `False` will skip the thermochemical analysis completely.

2.6 Type Aliases

Aliases are available for a large number of job types, allowing one to pass a `str` instead of a `type` object, thus simplifying the input settings for **CAT**. Aliases are insensitive towards capitalization (or lack thereof).

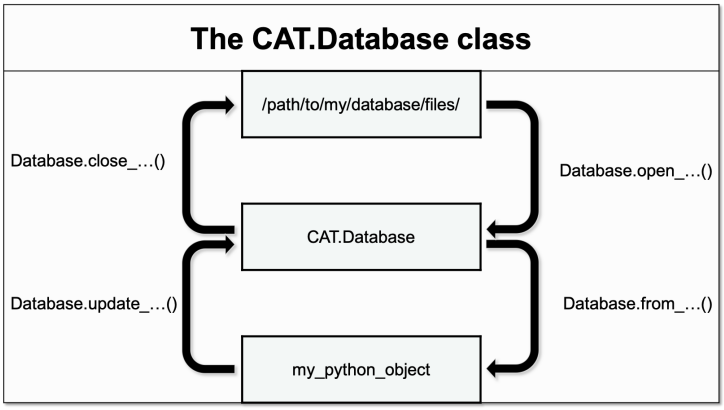
A comprehensive list of `plams.Job` subclasses and their respective aliases (*i.e.* `str`) is presented below.

2.6.1 Aliases

- ADFJob = "adf" = "adfjob"
- AMSJob = "ams" = "amsjob"
- UFFJob = "uff" = "uffjob"
- BANDJob = "band" = "bandjob"
- DFTBJob = "dftb" = "dftbjob"
- MOPACJob = "mopac" = "mopacjob"
- ReaxFFJob = "reaxff" = "reaxffjob"
- Cp2kJob = "cp2k" = "cp2kjob"
- ORCAJob = "orca" = "orcajob"
- DiracJob = "dirac" = "diracjob"
- GamessJob = "gamess" = "gamessjob"
- DFTBPlusJob = "dftbplus" = "dftbplusjob"
- CRSJob = "crs" = "cosmo-rs" = "crsjob"

2.7 The Database Class

A Class designed for the storing, retrieval and updating of results.



The methods of the Database class can be divided into three categories accoring to their functionality:

- Opening & closing the database - these methods serve as context managers for loading and unloading parts of the database from the harddrive.

  The context managers can be accessed via the *MetaManager.open()* method of *Database.csv_lig*, *Database.csv_qd*, *Database.yaml* or *Database.hdf5*, with the option of passing additional positional or keyword arguments.

```
>>> import CAT

>>> database = CAT.Database()
>>> with database.csv_lig.open(write=False) as db:
>>>     print(repr(db))
```

(continues on next page)

(continued from previous page)

```
DFCollection(df=<pandas.core.frame.DataFrame at 0x7ff8e958ce80>)

>>> with database.yaml.open() as db:
>>>     print(type(db))
<class 'scm.plams.core.settings.Settings'>

>>> with database.hdf5.open('r') as db:
>>>     print(type(db))
<class 'h5py._hl.files.File'>
```

- Importing to the database - these methods handle the importing of new data from python objects to the Database class:

update_csv()	*update_yaml()*	*update_hdf5()*	*update_mongodb()*

- Exporting from the database - these methods handle the exporting of data from the Database class to other python objects or remote locations:

from_csv()	*from_hdf5()*

2.7.1 Index

dirname	
csv_lig	
csv_qd	
hdf5	
yaml	
mongodb	
update_mongodb([database, overwrite])	Export ligand or qd results to the MongoDB database.
update_csv(df[, database, columns, ...])	Update `Database.csv_lig` or `Database.csv_qd` with new settings.
update_yaml(job_recipe)	Update `Database.yaml` with (potentially) new user provided settings.
update_hdf5(df[, database, overwrite, opt])	Export molecules (see the `"mol"` column in **df**) to the structure database.
from_csv(df[, database, get_mol, inplace])	Pull results from `Database.csv_lig` or `Database.csv_qd`.
from_hdf5(index[, database, rdmol])	Import structures from the hdf5 database as RDKit or PLAMS molecules.

df_collection.get_df_collection(df)	Return a mutable collection for holding dataframes.
database_functions.as_pdb_array(mol_list[, ...])	Convert a list of PLAMS molecule into an array of (partially) de-serialized .pdb files.
database_functions.from_pdb_array(array[, rdmol])	Convert an array with a (partially) de-serialized .pdb file into a molecule.
database_functions.sanitize_yaml_settings(...)	Remove a predetermined set of unwanted keys and values from a settings object.

2.7.2 Class API

Database

class dataCAT.database.**Database**(*path=None*, *host='localhost'*, *port=27017*, ***kwargs*)

The Database class.

Parameters

- **path** (*str*) – The path+directory name of the directory which is to contain all database components (see *Database.dirname*).
- **host** (*str*) – Hostname or IP address or Unix domain socket path of a single mongod or mongos instance to connect to, or a mongodb URI, or a list of hostnames mongodb URIs. If **host** is an IPv6 literal it must be enclosed in `"["` and `"]"` characters following the RFC2732 URL syntax (e.g. `"[::1]"` for localhost). Multihomed and round robin DNS addresses are not supported. See *Database.mongodb*.
- **port** (*str*) – port number on which to connect. See *Database.mongodb*.
- ****kwargs** – Optional keyword argument for pymongo.MongoClient. See *Database.mongodb*.

dirname

The path+filename of the directory containing all database components.

Type str

csv_lig

A dataclass for accesing the context manager for opening the .csv file containing all ligand related results.

Type *dataCAT.MetaManager*

csv_qd

A dataclass for accesing the context manager for opening the .csv file containing all quantum dot related results.

Type *dataCAT.MetaManager*

yaml

A dataclass for accesing the context manager for opening the .yaml file containing all job settings.

Type *dataCAT.MetaManager*

hdf5

A dataclass for accesing the context manager for opening the .hdf5 file containing all structures (as partiallize de-serialized .pdb files).

Type *dataCAT.MetaManager*

mongodb

Optional: A dictionary with keyword arguments for pymongo.MongoClient. Defaults to `None` if a `ServerSelectionTimeoutError` is raised when failing to contact the host. See the **host**, **port** and **kwargs** parameter.

Type dict

update_mongodb(*database='ligand'*, *overwrite=False*)

Export ligand or qd results to the MongoDB database.

Examples

```
>>> from CAT import Database

>>> db = Database(**kwargs)

# Update from db.csv_lig
>>> db.update_mongodb('ligand')

# Update from a lig_df, a user-provided DataFrame
>>> db.update_mongodb({'ligand': lig_df})
>>> print(type(lig_df))
<class 'pandas.core.frame.DataFrame'>
```

Parameters

- **database** (*str* or *dict* [*str*, *pd.DataFrame*]) – The type of database. Accepted values are `"ligand"` and `"QD"`, opening *Database.csv_lig* and *Database.csv_qd*, respectivelly. Alternativelly, a dictionary with the database name and a matching DataFrame can be passed directly.
- **overwrite** (*bool*) – Whether or not previous entries can be overwritten or not.

Return type `None`

update_csv(*df*, *database='ligand'*, *columns=None*, *overwrite=False*, *job_recipe=None*, *opt=False*)

Update *Database.csv_lig* or *Database.csv_qd* with new settings.

Parameters

- **df** (*pd.DataFrame*) – A dataframe of new (potential) database entries.
- **database** (*str*) – The type of database; accepted values are `"ligand"` (*Database.csv_lig*) and `"QD"` (*Database.csv_qd*).
- **columns** (*Sequence*) – Optional: A list of column keys in **df** which (potentially) are to be added to this instance. If `None`: Add all columns.
- **overwrite** (*bool*) – Whether or not previous entries can be overwritten or not.
- **job_recipe** (*plams.Settings*) – Optional: A `Settings` instance with settings specific to a job.
- **opt** (*bool*) – WiP.

Return type `None`

update_yaml(*job_recipe*)

Update *Database.yaml* with (potentially) new user provided settings.

Parameters **job_recipe** (*plams.Settings*) – A settings object with one or more settings specific to a job.

Returns A dictionary with the column names as keys and the key for *Database.yaml* as matching values.

Return type *dict_*

update_hdf5(*df*, *database='ligand'*, *overwrite=False*, *opt=False*)

Export molecules (see the `"mol"` column in **df**) to the structure database.

Returns a series with the *Database.hdf5* indices of all new entries.

Parameters

- **df** (*pd.DataFrame*) – A dataframe of new (potential) database entries.
- **database** (*str*) – The type of database; accepted values are `"ligand"` and `"QD"`.
- **overwrite** (*bool*) – Whether or not previous entries can be overwritten or not.

Returns A series with the indices of all new molecules in *Database.hdf5*.

Return type *pd.Series_*

from_csv(*df*, *database='ligand'*, *get_mol=True*, *inplace=True*)

Pull results from *Database.csv_lig* or *Database.csv_qd*.

Performs in inplace update of **df** if **inplace** = `True`, thus returing `None`.

Parameters

- **df** (*pd.DataFrame*) – A dataframe of new (potential) database entries.
- **database** (*str*) – The type of database; accepted values are `"ligand"` and `"QD"`.
- **get_mol** (*bool*) – Attempt to pull preexisting molecules from the database. See the **inplace** argument for more details.
- **inplace** (*bool*) – If `True` perform an inplace update of the `"mol"` column in **df**. Otherwise return a new series of PLAMS molecules.

Returns Optional: A Series of PLAMS molecules if **get_mol** = `True` and **inplace** = `False`.

Return type *pd.Series* [*plams.Molecule*]

from_hdf5(*index*, *database='ligand'*, *rdmol=True*)

Import structures from the hdf5 database as RDKit or PLAMS molecules.

Parameters

- **index** (*list* [*int*]) – The indices of the to be retrieved structures.
- **database** (*str*) – The type of database; accepted values are `"ligand"` and `"QD"`.
- **rdmol** (*bool*) – If `True`, return an RDKit molecule instead of a PLAMS molecule.
- **close** (*bool*) – If the database component (*Database.hdf5*) should be closed afterwards.

Returns A list of PLAMS or RDKit molecules.

Return type *list* [*plams.Molecule* or *rdkit.Chem.Mol*]

hdf5_availability(*timeout=5.0*, *max_attempts=None*)

Check if a .hdf5 file is opened by another process; return once it is not.

If two processes attempt to simultaneously open a single hdf5 file then h5py will raise an `OSError`.

The purpose of this method is ensure that a .hdf5 file is actually closed, thus allowing the *Database.from_hdf5()* method to safely access **filename** without the risk of raising an `OSError`.

Parameters

- **filename** (*str*) – The path+filename of the hdf5 file.
- **timeout** (*float*) – Time timeout, in seconds, between subsequent attempts of opening **filename**.
- **max_attempts** (*int*) – Optional: The maximum number attempts for opening **filename**. If the maximum number of attempts is exceeded, raise an `OSError`.

Raises **OSError** – Raised if **max_attempts** is exceded.

Return type None

DFCollection

class dataCAT.df_collection.**_DFCollection**(*df*)

A mutable collection for holding dataframes.

Parameters **df** (*pd.DataFrame*) – A Pandas DataFrame (see *_DFCollection.df*).

df

A Pandas DataFrame.

Type *pd.DataFrame*

> **Warning:** The *_DFCollection* class should never be directly called on its own. See *get_df_collection()*, which returns an actually usable DFCollection instance (a subclass).

MetaManager

class dataCAT.context_managers.**MetaManager**(*filename*, *manager*)

A wrapper for context managers.

Has a single important method, *MetaManager.open()*, which calls and returns the context manager stored in *MetaManager.manager*.

Note: *MetaManager.filename* will be the first positional argument provided to *MetaManager.manager*.

Parameters

- **filename** (*str*) – The path+filename of a database component See *MetaManager.filename*.
- **manager** (*type* [*AbstractContextManager*]) – A type object of a context manager. TThe first positional argument of the context manager should be the filename. See *MetaManager.manager*.

filename

The path+filename of a database component.

Type str

manager

A type object of a context manager. The first positional argument of the context manager should be the filename.

Type *type* [*AbstractContextManager*]

open(**args*, ***kwargs*)

Call and return *MetaManager.manager*.

Parameters

- ***args** – Positional arguments for *MetaManager.manager*.
- ****kwargs** – Keyword arguments for *MetaManager.manager*.

Returns An instance of a context manager.

Return type *AbstractContextManager_*

OpenLig

class dataCAT.context_managers.**OpenLig**(*filename=None*, *write=True*)

Context manager for opening and closing the ligand database (*Database.csv_lig*).

Parameters

- **filename** (*str*) – The path+filename to the database component.
- **write** (*bool*) – Whether or not the database file should be updated after closing this instance.

filename

The path+filename to the database component.

Type str

write

Whether or not the database file should be updated after closing this instance.

Type bool

df

An attribute for (temporary) storing the opened .csv file (see *OpenLig.filename*) as a DataFrame instance.

Type *None* or *pd.DataFrame*

OpenQD

class dataCAT.context_managers.**OpenQD**(*filename=None*, *write=True*)

Context manager for opening and closing the QD database (*Database.csv_qd*).

Parameters

- **filename** (*str*) – The path+filename to the database component.
- **write** (*bool*) – Whether or not the database file should be updated after closing this instance.

filename

The path+filename to the database component.

Type str

write

Whether or not the database file should be updated after closing this instance.

Type bool

df

An attribute for (temporary) storing the opened .csv file (*OpenQD.filename*) as DataFrame instance.

Type *None* or *pd.DataFrame*

OpenYaml

class dataCAT.context_managers.**OpenYaml**(*filename=None*, *write=True*)
Context manager for opening and closing job settings (*Database.yaml*).

Parameters

- **filename** (*str*) – The path+filename to the database component.
- **write** (*bool*) – Whether or not the database file should be updated after closing this instance.

filename
The path+filename to the database component.

Type str

write
Whether or not the database file should be updated after closing this instance.

Type bool

settings
An attribute for (temporary) storing the opened .yaml file (*OpenYaml.filename*) as Settings instance.

Type *None* or *plams.Settings*

2.7.3 Function API

dataCAT.df_collection.**get_df_collection**(*df*)
Return a mutable collection for holding dataframes.

Parameters **df** (*pd.DataFrame*) – A Pandas DataFrame.

Returns A DFCollection instance. The class is described in more detail in the documentation of its superclass: *_DFCollection*.

Return type *dataCAT.DFCollection_*

Note: As the DFCollection class is defined within the scope of this function, two instances of DFCollection will *not* belong to the same class (see example below). In more technical terms: The class bound to a particular DFCollection instance is a unique instance of type.

```
>>> import numpy as np
>>> import pandas as pd

>>> df = pd.DataFrame(np.random.rand(5, 5))
>>> collection1 = get_df_collection(df)
>>> collection2 = get_df_collection(df)

>>> print(df is collection1.df is collection2.df)
True

>>> print(collection1.__class__.__name__ == collection2.__class__.__name__)
True

>>> print(collection1.__class__ == collection2.__class__)
False
```

dataCAT.database_functions.**as_pdb_array**(*mol_list*, *min_size=0*)
Convert a list of PLAMS molecule into an array of (partially) de-serialized .pdb files.

Parameters

- **mol_list** (*m list* [*plams.Molecule*]) – A list of m PLAMS molecules.
- **min_size** (*int*) – The minimumum length of the pdb_array. The array is padded with empty strings if required.

Returns An array with m partially deserialized .pdb files with up to n lines each.

Return type $m * n$ *np.ndarray* [*np.bytes* *\S80*]

dataCAT.database_functions.**from_pdb_array**(*array*, *rdmol=True*)
Convert an array with a (partially) de-serialized .pdb file into a molecule.

Parameters

- **array** (*n np.ndarray* [*np.bytes* / S80]) – A (partially) de-serialized .pdb file with n lines.
- **rdmol** (*bool*) – If `True`, return an RDKit molecule instead of a PLAMS molecule.

Returns A PLAMS or RDKit molecule build from **array**.

Return type *plams.Molecule* or *rdkit.Chem.Mol_*

dataCAT.database_functions.**sanitize_yaml_settings**(*settings*, *job_type*)
Remove a predetermined set of unwanted keys and values from a settings object.

Parameters

- **settings** (*plams.Settings*) – A settings instance with, potentially, undesired keys and values.
- **job_type** (*str*) – The name of key in the settings blacklist.

Returns A new Settings instance with all unwanted keys and values removed.

Return type *plams.Settings_*

Raises **KeyError** – Raised if **jobtype** is not found in .../CAT/data/templates/settings_blacklist.yaml.

INDEX

Symbols

A

C

D

F

G

H

I

J

K

L

M