abcd Documentation

Release 0.1

libAtoms

April 01, 2016

Contents

Contents:

CHAPTER 1

Design Goals

1.1 Atom Based Computational Database

Provide the following:

- Command line tool to **store**, **interrogate** and **fetch** atomic configurations in a database.
- Python API to interact with the database in an analogous way to the CLI client.
- Backend specification so that the CLI and API can be interfaced with a wide range of database solutions.

Language and framework:

- Written in pure Python;
- Works flawlessly with Python 2.7 and 3.3 upwards;
- Depends on ASE for working with `Atoms` objects.

Backends:

- Agnostic according to defined specification.
- `ase.db` included
- 'mongodb included
- Aiida as a target

1.2 Design considerations

- Command line tool inspired by "icepick": store configurations, query, extract and update them and which is agnostic with respect to the back-end. At least two different back-ends will be created initially, one based on ase-db using James' patch, and Martin will make sure Aiida can also be used as a back-end.
- Communication between the command line tool and the backend is via ASE: files to be stored are read in via ASE's importers, and the Atoms object that is created (including all metadata) is passed to the backend. simple translators are written for Aiida using the already existing ASE importer (may need to be extended to pick up all metadata)
- The command line tool can be extended or built upon to do Chris's fetch-compute_property-store functionality, it is up to the database backend to tag the config with unique IDs so that subsequent stores are recognised as updates, we don't need to care about how that is done.

- queries: the command line tool needs to accept a set of predicates on the metadata. we can discuss and argue how general this needs to be: at the minimum, it is a list of predicates which are "and"-ed. the other end of the complexity is a complete predicate tree, allowing any combination of "and" and "or" relations between the predicates.
- Authentication: Martin says that Aiida is thinking about OpenID - I think in addition we need something much simpler as well, and there is no harm in multiple auth methods. I looked at how gitolite uses ssh keys, and it's simple: a single unix user is created on the system, and a number of keys can be placed in its .ssh/authorised_keys file. Each key in this file is associate with a command, e.g. "/usr/local/bin/abcd " and an argument to this command is the user name. The database is queried using ssh, e.g

```
ssh abcd@gc121mac1.eng.cam.ac.uk --command --line --arguments --and --query --predicates
```

  and when the user authenticates, instead of the shell, the /usr/local/bin/abcd command gets executed with the first argument being the and the subsequent arguments are taken from the above ssh command. So if I want to give someone access, all I have to do is to put their ssh key into this authorized_keys file. We can also permit anonymous access by having no password on this account, and the /usr/local/bin/abcd program would then execute without a argument, which would give access to those database objects that are tagged for anonymous access

CHAPTER 2

TODO

2.1 Frontend

- Create a UI for working with configuration files.
- Create a backend abstract factory
- Add general backend tests
- Add "interactive" mode to CLI (i.e. it doesn't auto return)
- Make the ASE install automatic (currently it asks the user to manually install the latest development version from https://wiki.fysik.dtu.dk/ase/download.html#latest-development-release)
- copy/move files from one database to another, including a new database
- Ability to add keys with commas
- Add the –unique option to the command line for the summary table

2.2 API

- Convert CLI into a Python class that can be interacted with using Python. CLI subcommands become methods.
- Relicense as LGPL?

2.3 asedb-based backend

- 'k!=v' looks for configurations containing a key "k" which is different from "v", instead of looking for all configurations for which !(k=v) evaluates to True (so configurations not containing "k" are not returned) - note this is an intended behaviour on the ASEdb end, not a bug.

2.4 mongodb-based backend

- Update it so it conforms to the Backend class

CHAPTER 3

API Documentation

3.1 abcd package

3.1.1 Submodules

3.1.2 abcd.authentication module

Classes related to facilitating authentication by the backend of some credentials gathered by the frontend.

class abcd.authentication.**AuthToken**(*username*)
Bases: object

username

exception abcd.authentication.**AuthenticationError**(*message*)
Bases: exceptions.Exception

class abcd.authentication.**Credentials**(*username=None*)
Bases: object

username
Get the username

Returns The username

class abcd.authentication.**UsernameAndPassword**(*username*, *password*)
Bases: *abcd.authentication.Credentials*

password

3.1.3 abcd.backend module

The backend interface that must be implemented by any structure storage library that wants to be compliant with this framework.

In general implementations of this class should perform translation from to commands understood by the native storage format being used be it SQL, a filesystem, MongoDB or others.

class abcd.backend.**Backend**
Bases: object

add_keys(*auth_token*, *filter*, *kvp*)
Adds key-value pairs to the selectd configurations

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **filter** (*dictionary?*) – Filter (in MongoDB query language)
- **kvp** (*dict*) – Key-value pairs to be added

Return type *AddKvpResult*

authenticate(*credentials*)
Take a set of credentials and return an authorisation token or raise an exception

Parameters **credentials** (Credentials) – The credentials, a subclass of

:py:class:Credentials :return: :rtype: AuthToken

close()

find(*auth_token*, *filter*, *sort*, *limit*, *keys*, *omit*)
Find entries that match the filter

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **filter** (*list of Conditions*) – Filter
- **sort** (*dict*) – Dictionary where keys are columns byt which to sort end values are either abcd.Direction.ASCENDING or abcd.Direction.DESCENDING
- **limit** (*int*) – limit the number of returned entries
- **keys** (list) – keys to be returned. None for all.
- **omit** (*bool*) – if True, the keys parameter will be interpreted as the keys to omit (all keys except the ones specified will be returned).

Returns

Return type Iterator to the Atoms object

insert(*auth_token*, *atoms*)
Take the Atoms object or an iterable to the Atoms and insert it to the database

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **atoms** (*Atoms or Atoms iterable*) – Atoms to insert

Returns Returns a result that holds a list of ids at which the objects were inserted and a message

Return type *InsertResult*

is_open()

list(*auth_token*)
List all the databases the user has access to

Parameters **auth_token** (AuthToken) – Authorisation token

Return type *list*

open()

remove(*auth_token*, *filter*, *just_one*)
Remove entries from the databse that match the filter

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **filter** (*dictionary?*) – Filter (in MongoDB query language)
- **just_one** (*bool*) – remove not more than one entry

Returns Returns a result that holds the number of removed entries and a message

Return type *RemoveResult*

remove_keys(*auth_token*, *filter*, *keys*)
Removes specified keys from selected configurations

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **filter** (*dictionary?*) – Filter (in MongoDB query language)
- **keys** (*dict*) – Keys to be removed

Return type *RemoveKeysResult*

update(*auth_token*, *atoms*, *upsert*, *replace*)
Take the atoms object and find an entry in the database with the same unique id. If one exists, the old entry gets updated with the new entry.

Parameters

- **auth_token** (AuthToken) – Authorisation token
- **atoms** (*Atoms or Atoms iterable*) – Atoms to insert
- **upsert** (*bool*) – Insert configurations even if they don't correspond to any existing ones
- **replace** (*bool*) – If a given configuration already exists, replace it

Returns

Return type *UpdateResult*

exception abcd.backend.**CommunicationError**(*message*)
Bases: exceptions.Exception

Error which is raised by the backend if communication with remote fails

class abcd.backend.**Cursor**
Bases: object

count()

next()

abcd.backend.**Direction**
alias of Enum

exception abcd.backend.**ReadError**(*message*)
Bases: exceptions.Exception

Error which is raised by the backend if read fails

exception abcd.backend.**WriteError**(*message*)
Bases: exceptions.Exception

Error which is raised by the backend if write fails

abcd.backend.**enum**(**sequential*)

3.1.4 abcd.cli module

abcd.cli.**main**()

abcd.cli.**print_result**(*result*, *multiconfig_files*, *database*)

abcd.cli.**run**(*args*, *sys_args*, *verbosity*)

abcd.cli.**to_stderr**(**args*)
Prints to stderr

abcd.cli.**untar_and_delete**(*tar_files*, *path_prefix*)

abcd.cli.**untar_file**(*fileobj*, *path_prefix*)

3.1.5 abcd.config module

config.py

Interact with configuration files and data files.

For testing, set XDG_CONFIG_HOME and XDG_DATA_HOME to avoid destroying existing files.

class abcd.config.**ConfigFile**(*module*, **args*, ***kwargs*)
Bases: ConfigParser.SafeConfigParser

Generic configuration file for specific parts of the code.

delete()

exists()
Return True if the config file already exists.

initialise(*data=None*, *overwrite=True*)
Create a new configuration file. If data is a dict the new configuration file will include the data as {section: {key: value}}

3.1.6 abcd.query module

exception abcd.query.**QueryError**(*message*)
Bases: exceptions.Exception

abcd.query.**elements2numbers**(*elements*)

abcd.query.**interpret**(*query*)
Translates a single query to the MongoDB format

abcd.query.**is_float**(*n*)

abcd.query.**is_int**(*n*)

abcd.query.**translate**(*queries_lst*)
Translates a list of queries to the MongoDB format

abcd.query.**update**(*d1*, *d2*)
Update dictionary d1 with d2

3.1.7 abcd.results module

class abcd.results.**AddKvpResult**(*modified_ids*, *no_of_kvp_added*, *msg=None*)
Bases: *abcd.results.Result*

modified_ids

no_of_kvp_added

class abcd.results.**InsertResult**(*inserted_ids*, *skipped_ids*, *msg=None*)
Bases: *abcd.results.Result*

inserted_ids

skipped_ids

class abcd.results.**RemoveKeysResult**(*modified_ids*, *no_of_keys_removed*, *msg=None*)
Bases: *abcd.results.Result*

modified_ids

no_of_keys_removed

class abcd.results.**RemoveResult**(*removed_count=1*, *msg=None*)
Bases: *abcd.results.Result*

removed_count
The number of entries removed :return: The number of entries removed

class abcd.results.**Result**(*msg=None*)
Bases: object

msg

class abcd.results.**UpdateResult**(*updated_ids*, *skipped_ids*, *upserted_ids*, *replaced_ids*, *msg=None*)
Bases: *abcd.results.Result*

replaced_ids

skipped_ids

updated_ids

upserted_ids

3.1.8 abcd.structurebox module

class abcd.structurebox.**StructureBox**(*backend*)
Bases: object

class BackendOpen(*backend*)

StructureBox.**add_keys**(*auth_token*, *filter*, *kvp*)

StructureBox.**authenticate**(*credentials*)

StructureBox.**find**(*auth_token*, *filter*, *sort={}*, *limit=0*, *keys=None*, *omit_keys=False*)

StructureBox.**insert**(*auth_token*, *atoms*)

StructureBox.**list**(*auth_token*)

StructureBox.**remove**(*auth_token*, *filter*, *just_one=True*)

StructureBox.**remove_keys**(*auth_token*, *filter*, *keys*)

StructureBox.**update**(*auth_token*, *atoms*, *upsert=False*, *replace=False*)

3.1.9 abcd.table module

abcd.table.**atoms_list2dict**(*atoms_it*)
Converts an Atoms iterator into a plain, one-level-deep list of dicts

abcd.table.**format_value**(*value*, *key*)
Applies special formatting for some key-value pairs

abcd.table.**print_keys_table**(*atoms_list*, *border=True*, *truncate=True*, *show_keys=[]*, *omit_keys=[]*)
Prints two tables: Intersection table and Union table, and shows min and max values for each key

abcd.table.**print_kvps**(*kvps*)
Takes a list of tuples, where each tuple is a key-value pair, and prints it.

abcd.table.**print_long_row**(*atoms*)
Prints full information about one configuration

abcd.table.**print_rows**(*atoms_list*, *border=True*, *truncate=True*, *show_keys=[]*, *omit_keys=[]*)
Prints a full table

abcd.table.**trim**(*val*, *length*)
Trim the string if it's longer than "length" (and add dots at the end)

3.1.10 abcd.util module

abcd.util.**atoms2dict**(*atoms*, *plain_arrays=False*)
Converts the Atoms object to a dictionary. If plain_arrays is True, numpy arrays are converted to lists.

abcd.util.**dict2atoms**(*d*, *plain_arrays=False*)
Converts a dictionary created with atoms2dict back to atoms.

abcd.util.**filter_keys**(*keys_list*, *keys*, *omit_keys*)
Decides which keys to show given keys and omit_keys

abcd.util.**get_info_and_arrays**(*atoms*, *plain_arrays*)
Extracts the info and arrays dictionaries from the Atoms object. If plain_arrays is True, numpy arrays are converted to lists.

CHAPTER 4

Backends

4.1 asedb_sqlite3_backend package

4.1.1 Submodules

4.1.2 asedb_sqlite3_backend.asedb_sqlite3_backend module

class asedb_sqlite3_backend.asedb_sqlite3_backend.**ASEdbSQlite3Backend**(*database=None*, *user=None*, *password=None*, *remote=None*)

Bases: *abcd.backend.Backend*

class **Cursor**(*iterator*)

Bases: *abcd.backend.Cursor*

count()

next()

ASEdbSQlite3Backend.**add_keys**(**args*, ***kwargs*)

ASEdbSQlite3Backend.**authenticate**(*credentials*)

ASEdbSQlite3Backend.**close**()

ASEdbSQlite3Backend.**connect_to_database**()

Connnects to a database with given name. If it doesn't exist, a new one is created. The method first looks in the "write" folder, and then in the "readonly" folder

ASEdbSQlite3Backend.**find**(**args*, ***kwargs*)

ASEdbSQlite3Backend.**insert**(**args*, ***kwargs*)

ASEdbSQlite3Backend.**is_open**()

ASEdbSQlite3Backend.**list**(*auth_token*)

ASEdbSQlite3Backend.**open**()

ASEdbSQlite3Backend.**read_only**(*func*)

ASEdbSQlite3Backend.**remove**(**args*, ***kwargs*)

ASEdbSQlite3Backend.**remove_keys**(**args*, ***kwargs*)

ASEdbSQlite3Backend.**require_database**(*func*)
When a function is decorated with this, an error will be thrown if the connection to a database is not open.

ASEdbSQlite3Backend.**update**(**args*, ***kwargs*)

asedb_sqlite3_backend.asedb_sqlite3_backend.**row2atoms**(*row*, *keys*, *omit_keys*)
keys: keys to show. None for all omit_keys: if true, all keys not in "keys" will be shown

4.1.3 asedb_sqlite3_backend.mongodb2asedb module

asedb_sqlite3_backend.mongodb2asedb.**interpret**(*key*, *op*, *val*)
Returns a list of ASEdb queries, where elements in this list are assumed to be ORed.

asedb_sqlite3_backend.mongodb2asedb.**translate_query**(*query*)
Translates the MongoDB query to the ASEdb query

4.1.4 asedb_sqlite3_backend.remote module

Functions that are used to communicate with a remote server (server.py).

asedb_sqlite3_backend.remote.**communicate_with_remote**(*host*, *command*)
Sends a command to the remote host and interprets and returns the response.

asedb_sqlite3_backend.remote.**result_from_dct**(*result_type*, ***kwargs*)
Re-creates a result that was converted to a dictionary.

4.1.5 asedb_sqlite3_backend.server module

Interface for the ASEdb backend. Its purpose is to be triggered by the communicate_with_remote function from remote.py, communicate with the ASEdb backend and print results/data to standard output. The output is b64-encoded and should be in a form XYZ:OUTPUT, where XYZ is the response code which indicates what type of output was produced (see below).

Response codes: 201: b64encoded string 202: json and b64encoded list 203: json and b64encoded dictionary 204: json and b64encoded list of dictionaries 220: json and b64encoded InsertResult dictionary 221: json and b64encoded UpdateResult dictionary 222: json and b64encoded RemoveResult dictionary 223: json and b64encoded AddKvpResult dictionary 224: json and b64encoded RemoveKeysResult dictionary 400: b64encoded string - Error 401: b64encoded string - ReadError 402: b64encoded string - WriteError

asedb_sqlite3_backend.server.**backendAddKeys**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendFind**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendInsert**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendList**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendRemove**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendRemoveKeys**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**backendUpdate**(**args*, ***kwargs*)

asedb_sqlite3_backend.server.**error_handler**(*func*)

asedb_sqlite3_backend.server.**main**()

4.1.6 asedb_sqlite3_backend.util module

asedb_sqlite3_backend.util.**add_user**(*user*)
Adds a user and their public key to ~/.ssh/authorized_keys file and creates directories $databases/USER and $databases/USER_readonly.

asedb_sqlite3_backend.util.**get_dbs_path**()
Reads the config file and returns the path to the folder where all the databases are stored.

asedb_sqlite3_backend.util.**main**()

asedb_sqlite3_backend.util.**print_usage**()

asedb_sqlite3_backend.util.**setup**()
Create a config file and a directory in which databases will be stored.

4.2 mongobackend package

4.2.1 Submodules

4.2.2 mongobackend.mongobackend module

class mongobackend.mongobackend.**MongoDBBackend**(*host*, *port*, *database='abcd'*, *collection='structures'*, *user=None*, *password=None*)
Bases: *abcd.backend.Backend*

class **Cursor**(*pymongo_cursor*)
Bases: *abcd.backend.Cursor*

count()

class MongoDBBackend.**Transform**
Bases: pymongo.son_manipulator.SONManipulator

transform_incoming(*son*, *collection*)

transform_outgoing(*son*, *collection*)

MongoDBBackend.**add_keys**(*auth_token*, *filter*, *kvp*)

MongoDBBackend.**authenticate**(*credentials*)

MongoDBBackend.**close**()

MongoDBBackend.**find**(*auth_token*, *filter*, *sort*, *reverse*, *limit*, *keys*, *omit_keys*)

MongoDBBackend.**insert**(*auth_token*, *atoms*, *kvp*)

MongoDBBackend.**is_open**()

MongoDBBackend.**list**(*auth_token*)

MongoDBBackend.**open**()

MongoDBBackend.**remove**(*auth_token*, *filter*, *just_one*, *confirm*)

MongoDBBackend.**remove_keys**(*auth_token*, *filter*, *keys*)

MongoDBBackend.**update**(*auth_token*, *atoms*)

CHAPTER 5

Indices and tables

- genindex
- modindex
- search

a

m

A

B

C

R

S

T

U

W