
malibu Documentation

Release 0.1.8-9-402be4c

Sean Johnson

Mar 22, 2017

Contents

1	Table of Contents	3
1.1	malibu API	3
1.1.1	malibu.command	3
1.1.1.1	malibu.command.module	4
1.1.2	malibu.config	6
1.1.2.1	malibu.config.configuration	7
1.1.3	malibu.database	8
1.1.3.1	malibu.database.dbmapper	9
1.1.3.2	Note from the author: (01 / 14 / 2016)	9
1.1.3.3	malibu.database.dbtypeconv	10
1.1.4	malibu.design	10
1.1.4.1	malibu.design.borgish	10
1.1.4.2	malibu.design.brine	11
1.1.5	malibu.text	14
1.1.6	malibu.util	14
2	Indices and tables	15
	Python Module Index	17

malibu is a collection of classes and utilities that make writing code a little bit easier and a little less tedious.

The whole point of this library is to have a small codebase that could be easily reused across projects with nice, easily loadable chunks that can be used disjointly.

Contents:

malibu API

malibu.command

Module for processing commands in a CLI fashion.

`malibu.command.get_command_modules(package=None)`

Reads a package and returns a dictionary of modules decorated with the `command_module()` decorator.

Parameters `package` (*str*) – Package to search for command modules

Returns dictionary of command modules

Return type dict

Raises `AttributeError` – if package has no `__all__` attribute

`malibu.command.command_module(func, *args, **kw)`

A decorator function that is used to register command modules in the `__command_modules` dictionary.

Parameters

- **func** (*function*) – Function being decorator
- ***args** – positional arguments
- ****kw** – keyword arguments

Returns none

Return type None

malibu.command.module

A relatively self-contained system for loading and creating command “modules” for a CLI-style script.

A command module must extend the *CommandModule* and set up the class as such:

```
from malibu.command import command_module, module

@command_module(
    name = "example",
    depends = []
)
class ExampleModule(module.CommandModule):

    def __init__(self, loader):

        super(ExampleModule, self).__init__()
        self.__loader = loader

        self.register_subcommand("help", self.show_help)

    def show_help(self, *args, **kw):
        """ example:help []

        Does something.
        """

        if "args" in kw:
            argparser = kw["args"]
        else:
            argparser = self.__loader.get_argument_parser()

        pass # Do stuff...
```

After all modules are implemented, use something similar to the following in a console entry point:

```
from malibu import command
from malibu.util import args

argparser = args.ArgumentParser.from_argv()
# Set up argparser params, mappings, etc here.

modloader = module.CommandModuleLoader(argparser)

mods = command.get_command_modules(package = __package__)
# Or replace __package__ with your cmd module package path
modloader.register_modules(mods.values())
modloader.instantiate_modules()

argparser.parse()

modloader.parse_command(
    argparser.parameters[1], # module base
    *argparser.parameters[1:], # subcommand and args
    args = argparser)
```

class malibu.command.module.CommandModuleLoader (*argparser, *args, **kw*)

Initializes a ModuleLoader object with a list for modules and sets the static `__instance` so the object is always accessible.

deinit_modules ()

Runs `__deinit__` on all registered modules.

deregister_module (*obj*)

Removes a module instance from the list of registered modules.

get_argument_parser ()

Returns the argument parser that will be passed into functions that are called by the loader as command line parameters. Allows modules to access the parser during instantiation to change param modules, add help text, register aliases, etc.

get_module_by_base (*modbase*)

Returns a module instance by the module's base name. Returns None if the named instance does not exist.

instantiate_modules (*clslist=[]*)

Instantiates all module classes that are registered. ** Might perform dependency lookup as well.

modules

Returns the list of modules.

parse_command (*command*, **args*, ***kw*)

Process a command and fire the function for the matching command and subcommand. Returns the function execution result, if any.

register_module (*cls*)

Registers a single module in the modloader list.

Checks if any module instances in the module list and raises if an instance already exists.

Otherwise, the module is instantiated, appended, and returned.

register_modules (*clslist*)

Registers a list of modules through subsequent calls to `register_module()`. Returns the list of instantiated modules.

class `malibu.command.module.CommandModule` (*base=None*)

Module superclass. Abstracts away some parts of a command module to make implementation simpler. Should only be inherited, never instantiated by itself.

Initializes a Module object with the command base, maps, and help dictionary.

execute_subcommand (*subcommand*, **args*, ***kw*)

Attempts to fire the subcommand with arguments, and keywords, may throw `CommandModuleException`.

get_base ()

Returns the command base.

get_help ()

Returns the help dictionary for this module.

has_alias (*alias*)

Boolean-returning method for if this Module has registered a specific alias.

has_subcommand (*subcommand*)

Boolean-returning method for if this Module has registered a specific subcommand.

is_command (*command*)

Simple boolean-returning method used during command parsing.

register_subcommand (*subcommand*, *function*, *aliases=[]*)

Registers a subcommand and its help in the internal maps. Updates aliases, subcommands, etc.

resolve_alias (*alias*)

Resolves an alias to a subcommand and returns the subcommand.

unregister_subcommand (*subcommand*)

Removes a subcommand, all aliases, and all help from the internal maps.

exception `malibu.command.module.CommandModuleException` (*value*)

Super special exception for modules.

malibu.config

Relatively simplistic configuration file loader, parser, and writer.

Has the ability to cross-link sections and load into completely de-serialized Python dictionaries.

Example of configuration format:

```
; filename: example.ini

[server]
address = 127.0.0.1
port = 1234

[default_route]
uri = /
controller = example_project.views.ExampleView

[static_content]
uri = /other_place
content = +url:https://maio.me/~pirogoeth/VIRUS.txt

[file_content]
uri = /another_place
content = +file:/var/data/content.txt

[routes]
routes = +list:["@default_route"]
```

Usage:

```
from malibu.config import configuration
conf = configuration.Configuration()
conf.load("example.ini")

# Grab a section
srv_opts = conf.get_section("server")

# Iterate over sections
for section in conf.sections:
    # do section...
    pass

# Add a section
data = {
    "value": "abc",
    "num": 123
}

conf.add_section("test", data)
conf.add_section("test2", data)
```

```
# Remove a section
conf.remove_section("test2")

# Save the changed configuration
conf.save()

# Or, save to a new file
conf.save(filename = "/var/data/config2.ini")
```

malibu.config.configuration

INI-style configuration implementation with some special features to make configuration a little simpler.

class malibu.config.configuration.Configuration

Configuration class performs the loading, saving, and parsing of an INI-style configuration file with a few advanced features such as value typing, file inclusion, section references, and JSON-style list definition.

initialise the container store in key:value format withing the certain category

add_section(section_name)

Adds a new configuration section to the main dictionary.

get_namespace(namespace)

Returns a set of ConfigurationSection objects that are prefixed with the namespace specified above.

If no configuration sections have the requested namespace, None is returned.

Parameters **namespace** (*str*) – Namespace to find in section name.

Return type *set*

Returns dict or None

get_section(section_name)

Return the internal ConfigurationSection representation of a set of configuration entries.

Parameters **section_name** (*str*) – Section name to retrieve.

Return type *malibu.config.configuration.ConfigurationSection*

Returns ConfigurationSection or None

has_section(section_name)

Return if this configuration has a section named :param section_name:.

load(filename)

Loads a INI-style configuration from the given filename. If the file can not be opened from :param filename:, a ValueError is raised. Upon any other error, the exception is simply raised to the top.

Raises

- **ValueError** – if no filename provided.
- **Exception** – upon other error

load_file(fobj)

Performs the full load of the configuration file from the underlying file object. If a file object is not passed in :param fobj:, TypeError is raised.

Raises **TypeError** – if :param fobj: is not a file type

reload()

Reload the configuration from the initially specified file

remove_section (*section_name*)

Removes a section from the main dictionary.

save (*filename=None*)

Write the loaded configuration into the file specified by :param filename: or to the initially specified filename.

All linked sections are flattened into SectionPromise instances and written to the configuration properly.

Raises ValueError – if no save filename available.

sections

Returns a list of all sections in the configuration.

unload ()

Unload an entire configuration

class malibu.config.configuration.**ConfigurationSection**

The ConfigurationSection class is a modified dictionary that provides “helpers” to grab a configuration entry in it’s correct “type” form.

get (*key, default=None*)

The bare “get” on the underlying dictionary that returns the configuration entry in whatever form it was parsed as, typically a string.

get_bool (*key, default=False*)

Attempts to safely fetch the value mapped to by :param key:. After successful retrieval, a conditional coercion to boolean is attempt. If the coercion to boolean fails, :param default: is returned.

get_int (*key, default=None*)

Attempts to fetch and intify the value mapped to by :param key:. If an error occurs while trying to intify the value, :param default: will be returned.

get_list (*key, delimiter=u',', strip=True, default=[]*)

Attempts to take a something-delimited string and “listify” it. If an error occurs while attempting to listify, :param default: will be returned.

get_string (*key, default=u''*)

Attempts to take the value stored and retrieve it safely as a string. If the value mapped to by :param key: is “!None”, the object returned is NoneType.

If an error occurs while trying to safely retrieve the string, :param default: is returned.

set (*key, value*)

Allows programmatic setting of configuration entries.

set_mutable (*mutable*)

Enforces immutability on a configuration section.

class malibu.config.configuration.**SectionPromise** (*config, section, key, link*)

this is a configuration section promise to make resolution of linked sections post-load easier.

resolve ()

Resolves a SectionPromise into the proper dictionary value.

malibu.database

malibu’s database classes were mainly an experiment with ORM tech using Python’s introspection capabilities, closures, and properties.

The actual ORM class exists as malibu.database.dbmapper.DBMapper and should be inherited to be used properly.

malibu.database.dbmapper.dbtypeconv is a stub module for installing adapters into the sqlite3 module.

As of the 0.1.6 release, the DBMapper and dbtypeconv are both deprecated in favour of external ORM projects with better compatibility.

malibu.database.dbmapper

This is a small, hackish ORM for SQLite3.

Note from the author: (01 / 14 / 2016)

I've got to be honest, this is probably the worst code I have ever written and read. At this point, this code is so difficult to maintain and keep up to date for 2/3 compat that it is almost not worth the work. Especially considering that there are things like Peewee, SQLAlchemy, etc, this is not worth using or maintaining.

From this point forward, I recommend using some other, cleaner, better maintained solution such as Peewee. This DBMapper code will no longer be maintained and will be deprecated starting with the 0.1.6 release. The code will be removed as the 1.0.0 release approaches. There may be plans to replace this with a SQLite adapter for the malibu.design.brine series of classes that behave similar to this, just without all the cruft.

```
class malibu.database.dbmapper.DBMapper (db, keys, keytypes, options={'uniqueIndices': set([]),
                                     'primaryIndex': 0, 'autoincrIndex': True, 'gen-
                                     FTSVTs': False})
```

This is code for a relatively small ORM for SQLite built on top of the python-sqlite3 module.

static connect_database (dbpath)

Connects to a database at 'dbpath' and installs the json type converter "middleware" into the database system.

classmethod find (**kw)

Searches for a set of records that match the query built by the contents of the kwargs and returns a filterable list of contextualized results that can be modified.

classmethod find_all ()

Finds all rows that belong to a table and returns a filterable list of contextualized results. Please note that the list that is returned can be empty, but it should never be none.

static get_default_options ()

Returns a deep copy of the default options dictionary for modification in subclasses.

classmethod join (cond, a, b)

DBMapper.join(cond => other table to join on a => left column to join b => right column to join)

Performs a sqlite join on two tables. Returns the join results in a filterable list.

classmethod load (**kw)

Loads a *single* row from the database and populates it into the context cls this method was called under.

If the database returns more than one row for the kwarg query, this method will only return the first result!

If you want a list of matching rows, use find() or search().

classmethod new (**kw)

Creates a new contextual instance and returns the object. Only parameters defined in the kwargs will be passed in to the record creation query, as there is no support for default values yet. (06/11/15)

classmethod search (param)

This function will return a list of results that match the given param for a full text query. The search parameter should be in the form of a sqlite full text query, as defined here:

http://www.sqlite.org/fts3.html#section_3

As an example, suppose your table looked like this:

id	name	description
1 2 3	linux freebsd windows	some magic daemonic magic tomfoolery

A full text query for “name:linux magic” would return the first row because the name is linux and the description contains “magic”. A full text query just for “description:magic” would return both rows one and two because the descriptions contain the word “magic”.

```
classmethod set_db_options (db, keys, ktypes, options={'uniqueIndices': set([]), 'primaryIndex':  
0, 'autoincrIndex': True, 'genFTSVTs': False})
```

DBMapper.set_db_options(db => database instance keys => list of keys ktypes => list of key types
options => options dictionary (optional))

Sets options for a subclasses DBMapper context.

malibu.database.dbtypeconv

This module contains small functions for installing and performing JSON conversion on data coming out from a SQLite database.

Pretty much useless-ish without DBMapper.

```
malibu.database.dbtypeconv.install_json_converter()  
    Installs a json object converter into the sqlite3 module for quick type conversions.
```

malibu.design

The classes in this package are essentially small design experiments. Both pieces have a little bit of history behind them and were/are intended to replace either a design methodology or another part of the library. You can read more about their “history” in the class packages.

malibu.design.borgish

Borgish was designed as a more extended implementation of Alex Martelli’s Borg design pattern, which aims to provide state consistency similar to a singleton design, but without the terribleness of singletons.

```
class malibu.design.borgish.SharedState (*args, **kw)
```

This class is for meta-class use as a state machine for persistence so we don’t use any singleton design.

The module is “Borg-ish”, as this implementation is loosely based on the Borg design pattern by Alex Martelli.

Calls the classes state dict initializer and loads initial state, if provided.

```
drop_state (state)
```

Drops the state specified from the class’ shared state dictionary.

Parameters state (str) – Name of state to drop.

Return type bool

Returns True if state was dropped, False otherwise.

load_state (*state*)

Loads state into the class, overwriting all data that was previously stored.

Parameters **state** (*str*) – Name of state to load.

Return type None

Returns None

Raises **NameError** – If the named state does not exist.

save_state (*state*)

Saves class state into a namespace on the class' shared state dict.

Parameters **state** (*str*) – Name of state to save.

Return type None

Returns None

Raises **NameError** – If the named state already exists.

malibu.design.brine

Brine is a play on Python's pickle module, which is used for serializing data. Brine is used for serialization as well, but into JSON, not a binary structure.

class malibu.design.brine.**BrineObject** (**args, **kw*)

This object is for use as a base class for other data. Essentially, it will expose a set of members that can be set and then squashed down to a JSON object through a call to `to_json`.

It can also be used as a meta-class for the base of a caching object model or other neat things.

as_dict ()

Returns the dictionary representation of the fields in this object.

Return type dict

Returns Current object in dictionary form

classmethod **by_dict** (*data, read_only=False, **kw*)

Creates a new instance with fields from the data parameter as long as they match what is in `_fields`.

Also does recursion on nested Brine objects.

Parameters

- **cls** (*class*) – BrineObject subclass
- **data** (*dict*) – Dictionary to use for fields
- **read_only** (*bool*) – Set object read-only
- ****kw** (*dict*) – BrineObject initializer options

Return type *BrineObject*

Returns BrineObject subclass instance

Raises **TypeError** – If data is not dict type

classmethod **by_json** (*data, read_only=False, **kw*)

Creates a new instance and calls `from_json` on the instance.

Will take kwargs and pass to the underlying instance initializer.

Parameters

- **cls** (*class*) – Class method is running on
- **data** (*str*) – JSON string to create object from
- **read_only** (*bool*) – Set object read-only
- ****kw** (*dict*) – Object initializer options

Return type cls

Returns new BrineObject subclass instance

from_dict (*data*)

Creates a new instance with fields from the data parameter as long as they match what is in `_fields`.

Also does recursion on nested Brine objects.

NOTE: Modifies the BrineObject *in-place*! If there are recursive objects also provided in the dictionary that are defined on the original object, they will also be modified *in-place*!

Keys prefixed by an underscore will be inserted into the object, but will not be tracked in `_fields` or `_special_fields`.

WARNING: This silently ignores “bad” fields.

Parameters **data** (*dict*) – Dictionary to use for fields

Return type None

Returns None

Raises **TypeError** – If data is not dict type

from_json (*data*)

Converts the JSON data back into an object, then loads the data into the model instance.

NOTE: This changes the current model *in-place*!

Parameters **data** (*str*) – JSON string to import

Return type None

Returns None

read_only ()

Set object as read-only. After an object is set read-only, it can not be unset as read-only.

Return type None

Returns None

to_json ()

Converts the object into JSON form. Simple, right?

Return type str

Returns Current object in JSON string form.

class malibu.design.brine.**CachingBrineObject** (**args, **kw*)

This is a magical class that performs the same function as the BrineObject, but it also adds object caching, searching, and fuzzy searching on the cache. Also provided is cached field invalidation / “dirtying”.

dirty_dict ()

Dumps a dictionary of dirty fields.

Return type dict

Returns Dictionary of all *dirty* values

dirty_json()

Dumps the dirty dictionary as JSON.

Return type str

Returns JSON dictionary of dirty values

classmethod fuzzy_search (*ignore_case=False*, ***kw*)

Performs a fuzzy search on the cache to find objects that have at least a diff ratio of FUZZ_RATIO.

Note that this can return more than one object and it may not be accurate. Time will tell.

Returns a list of matches ordered by likelihood of match.

Parameters

- **cls** (*class*) – Class to fuzzy search on
- **ignore_case** (*bool*) – Whether searching should ignore case
- ****kw** (*dict*) – Fields to search

Return type list

Returns List of matching CachingBrineObjects

classmethod search (*ignore_case=False*, ***kw*)

Searches through the cache to find objects with field that match those given by the ****kw**.

Note that this can return more than one object.

Parameters

- **ignore_case** (*bool*) – Should search ignore case?
- ****kw** (*dict*) – Fields to search

Return type list

Returns List of matching CachingBrineObjects

uncache()

Removes the object from the state cache forcibly.

Return type None

Returns None

unmark (**fields*)

Unmarks some field as dirty. Should only be called after the upstream is updated or only if you know what you're doing!

Parameters ***fields** (*list*) – Fields to unmark

Return type None

Returns None

malibu.design.brine.fuzzy_ratio (*a*, *b*)

Compares two values using the SequenceMatcher from difflib. Used for ~approximated~ fuzzy search.

Parameters

- **a** (*str*) – lhs string
- **b** (*str*) – rhs string

Return type int

Returns Integer ration of a \leq b

malibu.text

malibu.util

CHAPTER 2

Indices and tables

- `genindex`
- `modindex`
- `search`

m

- `malibu`, [3](#)
- `malibu.command`, [3](#)
- `malibu.command.module`, [3](#)
- `malibu.config`, [6](#)
- `malibu.config.configuration`, [7](#)
- `malibu.database`, [8](#)
- `malibu.database.dbmapper`, [9](#)
- `malibu.database.dbtypeconv`, [10](#)
- `malibu.design`, [10](#)
- `malibu.design.borgish`, [10](#)
- `malibu.design.brine`, [11](#)
- `malibu.text`, [14](#)
- `malibu.util`, [14](#)

A

`add_section()` (malibu.config.configuration.Configuration method), 7

`as_dict()` (malibu.design.brine.BrineObject method), 11

B

`BrineObject` (class in malibu.design.brine), 11

`by_dict()` (malibu.design.brine.BrineObject class method), 11

`by_json()` (malibu.design.brine.BrineObject class method), 11

C

`CachingBrineObject` (class in malibu.design.brine), 12

`command_module()` (in module malibu.command), 3

`CommandModule` (class in malibu.command.module), 5

`CommandModuleException`, 6

`CommandModuleLoader` (class in malibu.command.module), 4

`Configuration` (class in malibu.config.configuration), 7

`ConfigurationSection` (class in malibu.config.configuration), 8

`connect_database()` (malibu.database.dbmapper.DBMapper static method), 9

D

`DBMapper` (class in malibu.database.dbmapper), 9

`deinit_modules()` (malibu.command.module.CommandModuleLoader method), 4

`deregister_module()` (malibu.command.module.CommandModuleLoader method), 5

`dirty_dict()` (malibu.design.brine.CachingBrineObject method), 12

`dirty_json()` (malibu.design.brine.CachingBrineObject method), 12

`drop_state()` (malibu.design.borgish.SharedState method), 10

E

`execute_subcommand()` (malibu.command.module.CommandModule method), 5

F

`find()` (malibu.database.dbmapper.DBMapper class method), 9

`find_all()` (malibu.database.dbmapper.DBMapper class method), 9

`from_dict()` (malibu.design.brine.BrineObject method), 12

`from_json()` (malibu.design.brine.BrineObject method), 12

`fuzzy_ratio()` (in module malibu.design.brine), 13

`fuzzy_search()` (malibu.design.brine.CachingBrineObject class method), 13

G

`get()` (malibu.config.configuration.ConfigurationSection method), 8

`get_argument_parser()` (malibu.command.module.CommandModuleLoader method), 5

`get_base()` (malibu.command.module.CommandModule method), 5

`get_bool()` (malibu.config.configuration.ConfigurationSection method), 8

`get_command_modules()` (in module malibu.command), 3

`get_default_options()` (malibu.database.dbmapper.DBMapper static method), 9

`get_help()` (malibu.command.module.CommandModule method), 5

`get_int()` (malibu.config.configuration.ConfigurationSection method), 8

`get_list()` (malibu.config.configuration.ConfigurationSection method), 8

get_module_by_base() (malibu.command.module.CommandModuleLoader method), 5

get_namespace() (malibu.config.configuration.Configuration method), 7

get_section() (malibu.config.configuration.Configuration method), 7

get_string() (malibu.config.configuration.ConfigurationSection method), 8

malibu.design.brine (module), 11

malibu.text (module), 14

malibu.util (module), 14

modules (malibu.command.module.CommandModuleLoader attribute), 5

new() (malibu.database.dbmapper.DBMapper class method), 9

H

has_alias() (malibu.command.module.CommandModule method), 5

has_section() (malibu.config.configuration.Configuration method), 7

has_subcommand() (malibu.command.module.CommandModule method), 5

I

install_json_converter() (in module malibu.database.dbtypeconv), 10

instantiate_modules() (malibu.command.module.CommandModuleLoader method), 5

is_command() (malibu.command.module.CommandModule method), 5

J

join() (malibu.database.dbmapper.DBMapper class method), 9

L

load() (malibu.config.configuration.Configuration method), 7

load() (malibu.database.dbmapper.DBMapper class method), 9

load_file() (malibu.config.configuration.Configuration method), 7

load_state() (malibu.design.borgish.SharedState method), 10

M

malibu (module), 1

malibu.command (module), 3

malibu.command.module (module), 3

malibu.config (module), 6

malibu.config.configuration (module), 7

malibu.database (module), 8

malibu.database.dbmapper (module), 9

malibu.database.dbtypeconv (module), 10

malibu.design (module), 10

malibu.design.borgish (module), 10

N

P

parse_command() (malibu.command.module.CommandModuleLoader method), 5

R

read_only() (malibu.design.brine.BrineObject method), 12

register_module() (malibu.command.module.CommandModuleLoader method), 5

register_modules() (malibu.command.module.CommandModuleLoader method), 5

register_subcommand() (malibu.command.module.CommandModule method), 5

reload() (malibu.config.configuration.Configuration method), 7

remove_section() (malibu.config.configuration.Configuration method), 7

resolve() (malibu.config.configuration.SectionPromise method), 8

resolve_alias() (malibu.command.module.CommandModule method), 5

S

save() (malibu.config.configuration.Configuration method), 8

save_state() (malibu.design.borgish.SharedState method), 11

search() (malibu.database.dbmapper.DBMapper class method), 9

search() (malibu.design.brine.CachingBrineObject class method), 13

SectionPromise (class in malibu.config.configuration), 8

sections (malibu.config.configuration.Configuration attribute), 8

set() (malibu.config.configuration.ConfigurationSection method), 8

set_db_options() (malibu.database.dbmapper.DBMapper class method), 10

set_mutable() (malibu.config.configuration.ConfigurationSection
method), 8

SharedState (class in malibu.design.borgish), 10

T

to_json() (malibu.design.brine.BrineObject method), 12

U

uncache() (malibu.design.brine.CachingBrineObject
method), 13

unload() (malibu.config.configuration.Configuration
method), 8

unmark() (malibu.design.brine.CachingBrineObject
method), 13

unregister_subcommand() (mal-
ibu.command.module.CommandModule
method), 5