malibu Documentation

Release 0.1.5

Sean Johnson

January 14, 2016

Contents

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.

CHAPTER 1

Table of Contents

Contents:

1.1 malibu API

1.1.1 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):

    BASE = "example"

    def __init__(self, loader):

        super(ExampleModule, self).__init__(base = ExampleModule.BASE)
        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.

1.1.2 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_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.

1.1.3 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.

class malibu.database.dbmapper.**DBMapper**(*db*, *keys*, *keytypes*, *options={'uniqueIndices': set([])*, *'primaryIndex': 0*, *'autoincrIndex': True*, *'genFTSVTs': False}*)
This is code for a relatively small ORM for SQLite built on top of the python-sqlite3 module.

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.

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.**install_json_converter**()

Installs a json object converter into the sqlite3 module for quick type conversions.

1.1.4 malibu.design

1.1.5 malibu.text

1.1.6 malibu.util

CHAPTER 2

Indices and tables

- genindex
- modindex
- search

m

A

C

D

E

F

G

H

I