model-organization Documentation

Release 0.1.10

Philipp Sommer

Dec 27, 2018

Contents

Welcome! This package attempts to create an interface for managing the usage of a climate model. It provides the `ModelOrganization` class that manages different experiments in projects.

CHAPTER 1

Content

1.1 Getting started

1.1.1 Motivation

Developing computational models can be quite a mess in terms of the file and configuration management. Therefore most of the (climate) models are accompanied with some kind of framework to guide the user through their piece of software. Those however can be very complex and every model follows it's own strategy. Therefore this package tries to organize the procedure focusing at the end-user by providing an outer framework that allows the user to interfere with the model from the command line.

1.1.2 How does the package work

When doing research, we usually have a specific (funding) *project* that requests multiple runs (*experiments*) of our model. The framework of the model-organization package mirrors this basic structure: It works with projects, each project contains several experiments. The package keeps track of your projects and experiments and saves the configuration in separate configuration (.yml) files.

Configuration files

All the paths to the projects are stored in the configuration directory determined by the *name* attribute of your model (see the *config.get_configdir()* function). By default, it's (on linux and mac) "~/.config/<name>", but you can determine it via the global <NAME>CONFIGDIR variable, where <name> must be replaced by the *ModelOrganizer.name* of your model. Our *example below* would store the configuration files in "$HOME/.config/square" and the corresponding environment variable is SQUARECONFIGDIR.

The above directory contains 3 files:

globals.yml The global configuration that should be applicable to all projects

projects.yml A mapping from project name to project directory

experiments.yml The list of experiments

Additional, each project directory contains a `'.project'` directory where each experiment is represented by one yaml file (e.g. `'sine.yml'` in our *example*) and the project configuration is stored in `'.project/.project.yml'`. To get the specific name of the configuration file, you can also use the *exp_path* parameter of the *info()* method or the command *model info* respectively.

Creating your own `ModelOrganizer`

The *ModelOrganizer* class is designed for subclassing in order to fit to your specific model. See the incode documentation of the *ModelOrganizer* for more information.

Using the command line argument

Each method that is listed in the *commands* attribute is implemented as a subparser for the for the main command line utility (see our *Starting example: square*). If you subclassed the model organizer, you can use the *main()* method to run your model. You can do (for example) as we did in the *example* via:

```
if __name__ == '__main__':
    SquareModelOrganizer.main()
```

You can see, the results of this methodology in the *Command Line API Reference*.

1.1.3 Parallel usage

The usage of configuration files includes some limitations because the configuration cannot be accessed in parallel. Hence, you should not *setup projects* and *initialize experiments* in parallel. The same for the archiving method. However, the `run`, `postproc` and `preproc` method as we implemented in in our *example*, could be used in parallel.

1.2 Starting example: square

This small example will you demonstrate the basic methodology how our package works. We use a very simple model represented by only one function that squares the input data and we create a project to investigate trigonmetric functions (sine, cosine, etc.).

So let's define our model function

```
"""A small example for a computational model
"""


def compute(x):
    return x ** 2
```

and save it in a file called `'calculate.py'` file. This function runs only in python and does not know anything about where the input is from and where it is saved.

We start from the pure *ModelOrganizer* to manage our model named *square*

```
#!/usr/bin/env python
from model_organization import ModelOrganizer


class SquareModelOrganizer(ModelOrganizer):

    name = 'square'

if __name__ == '__main__':
    SquareModelOrganizer.main()
```

and save it in the file called `'square.py'`. If we run our model from the command line, (for technical reasons we run it here from inside IPython), we see already several preconfigured commands

```
In [1]: !./square.py -h
usage: square [-h] [-id str] [-l] [-n] [-v] [-vl str or int] [-nm] [-E]
              {setup,init,set-value,get-value,del-value,info,unarchive,configure,
↪archive,remove}
              ...

The main function for parsing global arguments

positional arguments:
  {setup,init,set-value,get-value,del-value,info,unarchive,configure,archive,remove}
    setup               Perform the initial setup for the project
    init                Initialize a new experiment
    set-value           Set a value in the configuration
    get-value           Get one or more values in the configuration
    del-value           Delete a value in the configuration
    info                Print information on the experiments
    unarchive           Extract archived experiments
    configure           Configure the project and experiments
    archive             Archive one or more experiments or a project instance
    remove              Delete an existing experiment and/or projectname

optional arguments:
  -h, --help            show this help message and exit
  -id str, --experiment str
                        experiment: str
                            The id of the experiment to use. If the `init` argument␣
↪is called, the `new` argument is automatically set. Otherwise, if not specified␣
↪differently, the last created experiment is used.
  -l, --last            If True, the last experiment is used
  -n, --new             If True, a new experiment is created
  -v, --verbose         Increase the verbosity level to DEBUG. See also `verbosity_
↪level`
                        for a more specific determination of the verbosity
  -vl str or int, --verbosity-level str or int
                        The verbosity level to use. Either one of ``'DEBUG', 'INFO',
                        'WARNING', 'ERROR'`` or the corresponding integer (see pythons
                        logging module)
  -nm, --no-modification
                        If True/set, no modifications in the configuration files will␣
↪be
                        done
  -E, --match           If True/set, interprete `experiment` as a regular expression
                        (regex) und use the matching experiment
```

So to setup our new project, we use the *setup()* method

```
In [2]: !./square.py setup . -p trigo
INFO:square.trigo:Initializing project trigo
```

And we initialize one experiment for the sine, one for the cosine, and one for the tangent functions

```
In [3]: !./square.py -id sine init -d "Squared Sine"
INFO:square.sine:Initializing experiment sine of project trigo

In [4]: !./square.py -id cosine init -d "Squared Cosine"
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\INFO:square.
↪cosine:Initializing experiment cosine of project trigo

In [5]: !./square.py -id tangent init -d "Squared Tangent"
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪tangent:Initializing experiment tangent of project trigo
```

Now of course, we, need the input data, so we enhance our *ModelOrganizer* subclass with a preprocess method

```
#!/usr/bin/env python
import os.path as osp
import numpy as np
from model_organization import ModelOrganizer


class SquareModelOrganizer(ModelOrganizer):

    name = 'square'

    commands = ModelOrganizer.commands[:]

    # insert the new run command to the other commands, right before the
    # archiving
    commands.insert(commands.index('archive'), 'preproc')

    def preproc(self, which='sin', **kwargs):
        """
        Create preprocessing data

        Parameters
        ----------
        which: str
            The name of the numpy function to apply
        ``**kwargs``
            Any other parameter for the
            :meth:`model_organization.ModelOrganizer.app_main` method
        """
        self.app_main(**kwargs)
        config = self.exp_config
        config['infile'] = infile = osp.join(config['expdir'], 'input.dat')

        func = getattr(np, which)  # np.sin, np.cos or np.tan
        data = func(np.linspace(-np.pi, np.pi))
        self.logger.info('Saving input data to %s', infile)
        np.savetxt(infile, data)

    def _modify_preproc(self, parser):
```

(continues on next page)

(continued from previous page)

```
        """Modify the parser for the preprocessing command"""
        parser.update_arg('which', short='w', choices=['sin', 'cos', 'tan'])

if __name__ == '__main__':
    SquareModelOrganizer.main()
```

The code should be more or less self explanatory. We start with the `ModelOrganizer.app_main()` method which chooses the right experiment from the given *id* parameter. The we use the `numpy.sin`, `numpy.cos` and `numpy.tan` functions from the numpy module and calculate the data from $-\pi$ to π.

Finally we store it to an input file inside the experiment directory.

Hint: If you want to see the configuration values in the experiment config, use the `info()` and the `get-value` commands

```
In [6]: !./square.py info
id: tangent
description: Squared Tangent
project: trigo
expdir: /home/docs/checkouts/readthedocs.org/user_builds/model-organization/checkouts/
↪stable/docs/trigo/experiments/tangent
timestamps:
  init: '2018-12-27 12:48:29.594300'
  setup: '2018-12-27 12:48:26.902886'

In [7]: !./square.py get-value expdir
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪home/docs/checkouts/readthedocs.org/user_builds/model-organization/checkouts/stable/
↪docs/trigo/experiments/tangent
```

In the `_modify_preproc` section, we update argument of the `--which` argument to include a shorter `-w` command and some choices (see the `funcargparse.FuncArgParser` class for more information). Note the style of the documentation of the `preproc` method. We follow the numpy documentation guidelines such that the `funcargparse.FuncArgParser` can extract the docstring. Our new command now has been translated to a command line argument:

```
In [8]: !./square.py preproc -h
\\\\\\\\\\\\\\\\\\\\usage: square preproc [-h] [-w str]

Create preprocessing data

optional arguments:
  -h, --help           show this help message and exit
  -w str, --which str  The name of the numpy function to apply
```

and we can use it to create our input data

```
In [9]: !./square.py -id sine preproc
INFO:square.sine:Saving input data to /home/docs/checkouts/readthedocs.org/user_
↪builds/model-organization/checkouts/stable/docs/trigo/experiments/sine/input.dat

In [10]: !./square.py -id cosine preproc -w cos
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪cosine:Saving input data to /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo/experiments/cosine/input.dat
```

(continues on next page)

(continued from previous page)

```
# by default, the last created experiment (tangent) is used so the -id
# argument is not necessary
In [11]: !./square.py preproc -w tan
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪tangent:Saving input data to /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo/experiments/tangent/input.dat
```

Finally, we include a `run` command that uses our input data of the given experiment and for our model

```
    def run(self, **kwargs):
        """
        Run the model

        Parameters
        ----------
        ``**kwargs``
            Any other parameter for the
            :meth:`model_organization.ModelOrganizer.app_main` method
        """
        from calculate import compute
        self.app_main(**kwargs)

        # get the default output name
        output = osp.join(self.exp_config['expdir'], 'output.dat')

        # save the paths in the configuration
        self.exp_config['output'] = output

        # run the model
        data = np.loadtxt(self.exp_config['infile'])
        out = compute(data)
        # save the output
        self.logger.info('Saving output data to %s', osp.relpath(output))
        np.savetxt(output, out)

        # store some additional information in the configuration of the
        # experiment
        self.exp_config['mean'] = mean = float(out.mean())
        self.exp_config['std'] = std = float(out.std())
        self.logger.debug('Mean: %s, Standard deviation: %s', mean, std)
```

and we run it

```
In [12]: !./square.py -v -id sine run
\\\\\\\\\\\\\\\\\\\\\INFO:square.sine:Saving output data to trigo/experiments/sine/
↪output.dat
DEBUG:square.sine:Mean: 0.49000000000000016, Standard deviation: 0.35693136595149494

In [13]: !./square.py -v -id cosine run
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪cosine:Saving output data to trigo/experiments/cosine/output.dat
DEBUG:square.cosine:Mean: 0.51, Standard deviation: 0.35693136595149494

In [14]: !./square.py -v run
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪tangent:Saving output data to trigo/experiments/tangent/output.dat
```

(continues on next page)

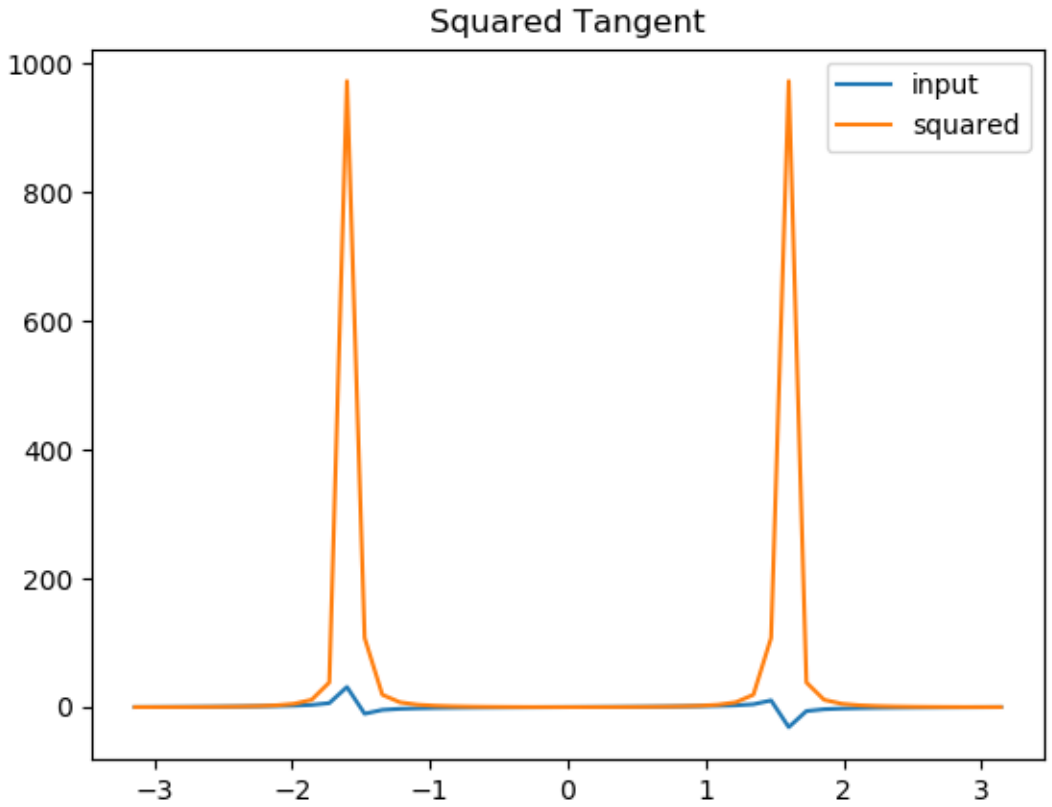
(continued from previous page)

```
DEBUG:square.tangent:Mean: 47.04000000000017, Standard deviation: 190.14783301421124
```

Now, finally, let's create a postproc command that visualizes our data

```
    def postproc(self, close=True, **kwargs):
        """
        Postprocess and visualize the data

        Parameters
        ----------
        close: bool
            Close the figure at the end
        ``**kwargs``
            Any other parameter for the
            :meth:`model_organization.ModelOrganizer.app_main` method
        """
        import matplotlib.pyplot as plt
        import seaborn as sns  # for nice plot styles
        self.app_main(**kwargs)

        # ---- load the data
        indata = np.loadtxt(self.exp_config['infile'])
        outdata = np.loadtxt(self.exp_config['output'])
        x_data = np.linspace(-np.pi, np.pi)

        # ---- make the plot
        fig = plt.figure()
        # plot input data
        plt.plot(x_data, indata, label='input')
        # plot output data
        plt.plot(x_data, outdata, label='squared')
        # draw a legend
        plt.legend()
        # use the description of the experiment as title
        plt.title(self.exp_config.get('description'))

        # ---- save the plot
        self.exp_config['plot'] = ofile = osp.join(self.exp_config['expdir'],
                                                   'plot.png')
        self.logger.info('Saving plot to %s', osp.relpath(ofile))
        fig.savefig(ofile)

        if close:
            plt.close(fig)
```

```
In [15]: !./square.py -v -id sine postproc
INFO:square.sine:Saving plot to trigo/experiments/sine/plot.png

In [16]: !./square.py -v -id cosine postproc
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\INFO:square.
↪cosine:Saving plot to trigo/experiments/cosine/plot.png

In [17]: !./square.py -v postproc
\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\\
↪tangent:Saving plot to trigo/experiments/tangent/plot.png
```



Finally, we can archive our project to save disc space

```
In [18]: !./square.py archive -p trigo -rm
INFO:square:Archiving to /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo.tar
```

Hint: You can also do everything from above in one line by chaining the subparser commands.

```
In [19]: !./square.py -v -id sine setup . -p trigo init -d "Squared Sine" preproc run␣
↪postproc archive -p trigo -rm
INFO:square.sine:Initializing project trigo
DEBUG:square.sine:    Creating root directory /home/docs/checkouts/readthedocs.org/
↪user_builds/model-organization/checkouts/stable/docs/trigo
INFO:square.sine:Initializing experiment sine of project trigo
DEBUG:square.sine:    Creating experiment directory /home/docs/checkouts/readthedocs.
↪org/user_builds/model-organization/checkouts/stable/docs/trigo/experiments/sine
INFO:square.sine:Saving input data to trigo/experiments/sine/input.dat
INFO:square.sine:Saving output data to trigo/experiments/sine/output.dat
DEBUG:square.sine:Mean: 0.49000000000000016, Standard deviation: 0.35693136595149494
INFO:square.sine:Saving plot to trigo/experiments/sine/plot.png
INFO:square.sine:Archiving to /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo.tar
DEBUG:square.sine:Adding /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo/experiments/sine
DEBUG:square.sine:Store sine experiment config to /home/docs/checkouts/readthedocs.
↪org/user_builds/model-organization/checkouts/stable/docs/trigo/.project/sine.yml
```

(continues on next page)

(continued from previous page)

```
DEBUG:square.sine:Store trigo project config to /home/docs/checkouts/readthedocs.org/
↪user_builds/model-organization/checkouts/stable/docs/trigo/.project/.project.yml
DEBUG:square.sine:Add /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo/.project to archive
DEBUG:square.sine:Removing /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo/experiments/sine
DEBUG:square.sine:Removing /home/docs/checkouts/readthedocs.org/user_builds/model-
↪organization/checkouts/stable/docs/trigo
```

1.3 API Reference

Classes

ModelOrganizer([config])	A class for organizing a model

class ModelOrganizer(*config=None*)

Bases: object

A class for organizing a model

This class is indended to hold the basic methods for organizing a model and should be adapted to your specific needs. The important steps are

1. specify the name of your model in the *name* attribute
2. use the existing commands, e.g. the *setup()* method or *init()* method if you need to adapt them to your specific needs
3. when you need your own commands,
   a. create a new method
   b. insert the name of the method in the *commands* attribute
   c. If necessary, you can use another name for the command for the command line parser and specify this one in the *parser_commands* attribute
   d. Create a _modify_<command> method (where <command> should be replaced by the name of the command) which accepts a funcargparse.FuncArgParser as an argument to update the parameters for the command line parsing
   e. In your new command, call the *app_main()* method which sets the correct experiment, etc.. You can also use the *get_app_main_kwargs()* method to filter out the right keyword arguments
4. When storing paths in the experiment configuration, we store them relative the project directory. In that way, we only have to modify the project configuration if we move our files to another place. Hence, if you want to use this possibility, you should include the configuration keys that represent file names in the *paths* attribute.

Path management

abspath(path[, project, root])	Returns the path from the current working directory
fix_paths(*args, **kwargs)	Fix the paths in the given dictionary to get absolute paths

Continued on next page

Table 2 – continued from previous page

paths	list of str. The keys describing paths for the model
rel_paths(*args, **kwargs)	Fix the paths in the given dictionary to get relative paths
relpath(path[, project, root])	Returns the relative path from the root directory of the project

Main functionality

app_main([experiment, last, new, verbose, ...])	The main function for parsing global arguments
experiment	The identifier or the experiment that is currently processed
is_archived(experiment[, ignore_missing])	Convenience function to determine whether the given experiment has been
main([args])	Run the organizer from the command line
parse_args([args])	Parse the arguments from the command line (or directly) to the parser
projectname	The name of the project that is currently processed
start(**kwargs)	Start the commands of this organizer

Infrastructural methods

archive([odir, aname, fmt, projectname, ...])	Archive one or more experiments or a project instance
init([projectname, description])	Initialize a new experiment
remove([projectname, complete, yes, ...])	Delete an existing experiment and/or projectname
setup(root_dir[, projectname, link])	Perform the initial setup for the project
unarchive([experiments, archive, complete, ...])	Extract archived experiments

Attributes

commands	commands that should be accessible from the command line. Each command
exp_config	The configuration settings of the current experiment
global_config	The global configuration settings
logger	The logger of this organizer
name	the name of the application/model
no_modification	bool(x) -> bool
project_config	The configuration settings of the current project of the

Configuration methods

configure([global_config, project_config, ...])	Configure the project and experiments
set_value(items[, complete, on_projects, ...])	Set a value in the configuration

Info methods

del_value(keys[, complete, on_projects, ...])	Delete a value in the configuration
get_value(keys[, exp_path, project_path, ...])	Get one or more values in the configuration

Continued on next page

Table 7 – continued from previous page

info([exp_path, project_path, global_path, ...])	Print information on the experiments

Methods

get_app_main_kwargs(kwargs[, keep])	Extract the keyword arguments for the `app_main()` method

Parser management

get_parser()	Function returning the command line parser for this class
parser	The `funcargparse.FuncArgParser` to use for controlling the model from the command line.
parser_commands	mapping from the name of the parser command to the method name
setup_parser([parser, subparsers])	Create the argument parser for this instance

Parameters **config** (`model_organization.config.Config`) – The configuration of the organizer

abspath(*path*, *project=None*, *root=None*)
Returns the path from the current working directory

We only store the paths relative to the root directory of the project. This method fixes those path to be applicable from the working directory

Parameters

- **path** (*str*) – The original path as it is stored in the configuration
- **project** (*str*) – The project to use. If None, the *projectname* attribute is used
- **root** (*str*) – If not None, the root directory of the project

Returns The path as it is accessible from the current working directory

Return type str

app_main(*experiment=None*, *last=False*, *new=False*, *verbose=False*, *verbosity_level=None*, *no_modification=False*, *match=False*)
The main function for parsing global arguments

Parameters

- **experiment** (*str*) – The id of the experiment to use
- **last** (*bool*) – If True, the last experiment is used
- **new** (*bool*) – If True, a new experiment is created
- **verbose** (*bool*) – Increase the verbosity level to DEBUG. See also *verbosity_level* for a more specific determination of the verbosity
- **verbosity_level** (*str or int*) – The verbosity level to use. Either one of `'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'` or the corresponding integer (see pythons logging module)
- **no_modification** (*bool*) – If True/set, no modifications in the configuration files will be done

- **match** (*bool*) – If True/set, interprete *experiment* as a regular expression (regex) und use the matching experiment

archive(*odir=None*, *aname=None*, *fmt=None*, *projectname=None*, *experiments=None*, *current_project=False*, *no_append=False*, *no_project_paths=False*, *exclude=None*, *keep_exp=False*, *rm_project=False*, *dry_run=False*, *dereference=False*, ***kwargs*)

Archive one or more experiments or a project instance

This method may be used to archive experiments in order to minimize the amount of necessary configuration files

Parameters

- **odir** (*str*) – The path where to store the archive
- **aname** (*str*) – The name of the archive (minus any format-specific extension). If None, defaults to the projectname
- **fmt** (*{ 'gztar' | 'bztar' | 'tar' | 'zip' }*) – The format of the archive. If None, it is tested whether an archived with the name specified by *aname* already exists and if yes, the format is inferred, otherwise `'tar'` is used
- **projectname** (*str*) – If provided, the entire project is archived
- **experiments** (*str*) – If provided, the given experiments are archived. Note that an error is raised if they belong to multiple project instances
- **current_project** (*bool*) – If True, *projectname* is set to the current project
- **no_append** (*bool*) – It True and the archive already exists, it is deleted
- **no_project_paths** (*bool*) – If True, paths outside the experiment directories are neglected
- **exclude** (*list of str*) – Filename patterns to ignore (see `glob.fnmatch.fnmatch()`)
- **keep_exp** (*bool*) – If True, the experiment directories are not removed and no modification is made in the configuration
- **rm_project** (*bool*) – If True, remove all the project files
- **dry_run** (*bool*) – If True, set, do not actually make anything
- **dereference** (*bool*) – If set, dereference symbolic links. Note: This is automatically set for `fmt=='zip'`

commands = ['setup', 'init', 'set_value', 'get_value', 'del_value', 'info', 'unarchive

commands that should be accessible from the command line. Each command corresponds to one method of this class

configure(*global_config=False*, *project_config=False*, *ifile=None*, *forcing=None*, *serial=False*, *nprocs=None*, *update_from=None*, ***kwargs*)

Configure the project and experiments

Parameters

- **global_config** (*bool*) – If True/set, the configuration are applied globally (already existing and configured experiments are not impacted)
- **project_config** (*bool*) – Apply the configuration on the entire project instance instead of only the single experiment (already existing and configured experiments are not impacted)
- **ifile** (*str*) – The input file for the project. Must be a netCDF file with population data

- **forcing** (*str*) – The input file for the project containing variables with population evolution information. Possible variables in the netCDF file are *movement* containing the number of people to move and *change* containing the population change (positive or negative)
- **serial** (*bool*) – Do the parameterization always serial (i.e. not in parallel on multiple processors). Does automatically impact global settings
- **nprocs** (*int or 'all'*) – Maximum number of processes to when making the parameterization in parallel. Does automatically impact global settings and disables *serial*
- **update_from** (*str*) – Path to a yaml configuration file to update the specified configuration with it
- ****kwargs** – Other keywords for the *app_main()* method

del_value(*keys*, *complete=False*, *on_projects=False*, *on_globals=False*, *projectname=None*, *base=''*, *dtype=None*, ***kwargs*)

Delete a value in the configuration

Parameters

- **keys** (*list of str*) – A list of keys to be deleted. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.
- **complete** (*bool*) – If True/set, the information on all experiments are printed
- **on_projects** (*bool*) – If set, show information on the projects rather than the experiment
- **on_globals** (*bool*) – If set, show the global configuration settings
- **projectname** (*str*) – The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown
- **base** (*str*) – A base string that shall be put in front of each key in *values* to avoid typing it all the time

exp_config

The configuration settings of the current experiment

experiment

The identifier or the experiment that is currently processed

fix_paths(**args*, ***kwargs*)

Fix the paths in the given dictionary to get absolute paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project
- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

get_app_main_kwargs(*kwargs*, *keep=False*)
Extract the keyword arguments for the *app_main()* method

Parameters

- **kwargs** (*dict*) – A mapping containing keyword arguments for the *app_main()* method
- **keep** (*bool*) – If True, the keywords are kept in the *kwargs*. Otherwise, they are removed

Returns The keyword arguments for the *app_main()* method

Return type dict

Notes

The returned keyword arguments are deleted from *kwargs*

classmethod get_parser()
Function returning the command line parser for this class

get_value(*keys*, *exp_path=False*, *project_path=False*, *complete=False*, *on_projects=False*, *on_globals=False*, *projectname=None*, *no_fix=False*, *only_keys=False*, *base=''*, *return_list=False*, *archives=False*, ***kwargs*)
Get one or more values in the configuration

Parameters

- **keys** (*list of str*) – A list of keys to get the values of. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.
- **exp_path** (*bool*) – If True/set, print the filename of the experiment configuration
- **project_path** (*bool*) – If True/set, print the filename on the project configuration
- **complete** (*bool*) – If True/set, the information on all experiments are printed
- **on_projects** (*bool*) – If set, show information on the projects rather than the experiment
- **on_globals** (*bool*) – If set, show the global configuration settings
- **projectname** (*str*) – The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown
- **no_fix** (*bool*) – If set, paths are given relative to the root directory of the project
- **only_keys** (*bool*) – If True, only the keys of the given dictionary are printed
- **archives** (*bool*) – If True, print the archives and the corresponding experiments for the specified project
- **base** (*str*) – A base string that shall be put in front of each key in *values* to avoid typing it all the time
- **return_list** (*bool*) – If True, the list of values corresponding to *keys* is returned, otherwise they are printed separated by a new line to the standard output

global_config
The global configuration settings

info(*exp_path=False*, *project_path=False*, *global_path=False*, *config_path=False*, *complete=False*, *no_fix=False*, *on_projects=False*, *on_globals=False*, *projectname=None*, *return_dict=False*, *insert_id=True*, *only_keys=False*, *archives=False*, ***kwargs*)
Print information on the experiments

Parameters

- **exp_path** (*bool*) – If True/set, print the filename of the experiment configuration
- **project_path** (*bool*) – If True/set, print the filename on the project configuration
- **global_path** (*bool*) – If True/set, print the filename on the global configuration
- **config_path** (*bool*) – If True/set, print the path to the configuration directory
- **complete** (*bool*) – If True/set, the information on all experiments are printed
- **no_fix** (*bool*) – If set, paths are given relative to the root directory of the project
- **on_projects** (*bool*) – If set, show information on the projects rather than the experiment
- **on_globals** (*bool*) – If set, show the global configuration settings
- **projectname** (*str*) – The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown
- **return_dict** (*bool*) – If True, the dictionary is returned instead of printed
- **insert_id** (*bool*) – If True and neither *on_projects*, nor *on_globals*, nor *projectname* is given, the experiment id is inserted in the dictionary
- **only_keys** (*bool*) – If True, only the keys of the given dictionary are printed
- **archives** (*bool*) – If True, print the archives and the corresponding experiments for the specified project

init(*projectname=None*, *description=None*, ***kwargs*)
Initialize a new experiment

Parameters

- **projectname** (*str*) – The name of the project that shall be used. If None, the last one created will be used
- **description** (*str*) – A short summary of the experiment
- ****kwargs** – Keyword arguments passed to the *app_main()* method

Notes

If the experiment is None, a new experiment will be created

is_archived(*experiment*, *ignore_missing=True*)
Convenience function to determine whether the given experiment has been archived already

Parameters **experiment** (*str*) – The experiment to check

Returns The path to the archive if it has been archived, otherwise None

Return type str or None

logger
The logger of this organizer

classmethod main(*args=None*)
Run the organizer from the command line

Parameters

- **name** (*str*) – The name of the program
- **args** (*list*) – The arguments that are parsed to the argument parser

name = 'model_organizer'
the name of the application/model

no_modification = False

parse_args(*args=None*)
Parse the arguments from the command line (or directly) to the parser of this organizer

Parameters **args** (*list*) – A list of arguments to parse. If None, the `sys.argv` argument is used

Returns The namespace with the commands as given in `**kwargs` and the return values of the corresponding method

Return type argparse.Namespace

parser = None
The `funcargparse.FuncArgParser` to use for controlling the model from the command line. This attribute is set by the *setup_parser()* method and used by the *start* method

parser_commands = {}
mapping from the name of the parser command to the method name

paths = ['expdir', 'src', 'data', 'input', 'outdata', 'outdir', 'plot_output', 'projec
list of str. The keys describing paths for the model

print_ = None

project_config
The configuration settings of the current project of the experiment

projectname
The name of the project that is currently processed

rel_paths(**args*, ***kwargs*)
Fix the paths in the given dictionary to get relative paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project
- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

relpath (*path*, *project=None*, *root=None*)
Returns the relative path from the root directory of the project

We only store the paths relative to the root directory of the project. This method gives you this path from a path that is accessible from the current working directory

Parameters

- **path** (*str*) – The original path accessible from the current working directory
- **project** (*str*) – The project to use. If None, the *projectname* attribute is used
- **root** (*str*) – If not None, the root directory of the project

Returns The path relative from the root directory

Return type str

remove (*projectname=None*, *complete=False*, *yes=False*, *all_projects=False*, ***kwargs*)
Delete an existing experiment and/or projectname

Parameters

- **projectname** (*str*) – The name for which the data shall be removed. If True, the project will be determined by the experiment. If not None, all experiments for the given project will be removed.
- **complete** (*bool*) – If set, delete not only the experiments and config files, but also all the project files
- **yes** (*bool*) – If True/set, do not ask for confirmation
- **all_projects** (*bool*) – If True/set, all projects are removed

Warning: This will remove the entire folder and all the related informations in the configurations!

set_value (*items*, *complete=False*, *on_projects=False*, *on_globals=False*, *projectname=None*, *base=''*, *dtype=None*, ***kwargs*)
Set a value in the configuration

Parameters

- **items** (*dict*) – A dictionary whose keys correspond to the item in the configuration and whose values are what shall be inserted. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.
- **complete** (*bool*) – If True/set, the information on all experiments are printed
- **on_projects** (*bool*) – If set, show information on the projects rather than the experiment
- **on_globals** (*bool*) – If set, show the global configuration settings
- **projectname** (*str*) – The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown
- **base** (*str*) – A base string that shall be put in front of each key in *values* to avoid typing it all the time
- **dtype** (*str*) – The name of the data type or a data type to cast the value to

setup (*root_dir*, *projectname=None*, *link=False*, ***kwargs*)
Perform the initial setup for the project

Parameters

- **root_dir** (*str*) – The path to the root directory where the experiments, etc. will be stored
- **projectname** (*str*) – The name of the project that shall be initialized at *root_dir*. A new directory will be created namely `root_dir + '/' + projectname`
- **link** (*bool*) – If set, the source files are linked to the original ones instead of copied

Other Parameters ‘‘**kwargs‘‘ – Are passed to the *app_main()* method

setup_parser(*parser=None*, *subparsers=None*)

Create the argument parser for this instance

This method uses the functions defined in the *commands* attribute to create a command line utility via the `FuncArgParser` class. Each command in the *commands* attribute is interpreted as on subparser and setup initially via the `FuncArgParser.setup_args()` method. You can modify the parser for each command *cmd* by including a `_modify_cmd` method that accepts the subparser as an argument

Parameters

- **parser** (*FuncArgParser*) – The parser to use. If None, a new one will be created
- **subparsers** (*argparse._SubParsersAction*) – The subparsers to use. If None, the `add_subparser` method from *parser* will be called

Returns

- *FuncArgParser* – The created command line parser or the given *parser*
- *argparse._SubParsersAction* – The created subparsers action or the given *subparsers*
- *dict* – A mapping from command name in the *commands* attribute to the corresponding command line parser

See also:

parse_args()

start(***kwargs*)

Start the commands of this organizer

Parameters ****kwargs** – Any keyword from the *commands* or *parser_commands* attribute

Returns The namespace with the commands as given in `**kwargs` and the return values of the corresponding method

Return type argparse.Namespace

unarchive(*experiments=None*, *archive=None*, *complete=False*, *project_data=False*, *replace_project_config=False*, *root=None*, *projectname=None*, *fmt=None*, *force=False*, ***kwargs*)

Extract archived experiments

Parameters

- **experiments** (*list of str*) – The experiments to extract. If None the current experiment is used
- **archive** (*str*) – The path to an archive to extract the experiments from. If None, we assume that the path to the archive has been stored in the configuration when using the *archive()* command

- **complete** (*bool*) – If True, archives are extracted completely, not only the experiment (implies `project_data = True`)
- **project_data** (*bool*) – If True, the data for the project is extracted as well
- **replace_project_config** (*bool*) – If True and the project does already exist in the configuration, it is updated with what is stored in the archive
- **root** (*str*) – An alternative root directory to use. Otherwise the experiment will be exctracted to
  1. the root directory specified in the configuration files (if the project exists in it) and *replace_project_config* is False
  2. the root directory as stored in the archive
- **projectname** (*str*) – The projectname to use. If None, use the one specified in the archive
- **fmt** (*{ 'gztar' | 'bztar' | 'tar' | 'zip' }*) – The format of the archive. If None, it is inferred
- **force** (*bool*) – If True, force to overwrite the configuration of all experiments from what is stored in *archive*. Otherwise, the configuration of the experiments in *archive* are only used if missing in the current configuration

1.3.1 Submodules

model_organization.config module

Classes

Archive	Just a dummy string subclass to identify archived experiments
Config(name)	Configuration class for one model organizer
ExperimentsConfig(projects[, d, project_map])	The configuration of the experiments
ProjectsConfig(conf_dir[, d])	The project configuration

Functions

get_configdir(name)	Return the string representing the configuration directory.
ordered_yaml_dump(data[, stream, Dumper])	Dumps the stream from an OrderedDict.
ordered_yaml_load(stream[, Loader, ...])	Loads the stream into an OrderedDict.
safe_dump(d, fname, *args, **kwargs)	Savely dump *d* to *fname* using yaml
safe_load(fname)	Load the file fname and make sure it can be done in parallel
setup_logging([default_path, default_level, ...])	Setup logging configuration

class Archive

Bases: `str`

Just a dummy string subclass to identify archived experiments **Attributes**

project	The name of the project inside this archive
time	The time when this project has been archived

project = None
The name of the project inside this archive

time = None
The time when this project has been archived

class Config(*name*)
Bases: object

Configuration class for one model organizer **Attributes**

experiments	ExperimentConfig. The configuration of the experiments
global_config	OrderedDict. The global configuration that applies to all
projects	*ProjectsConfig*. The configuration of the projects

Methods

remove_experiment(experiment)	
save()	Save the entire configuration files

experiments = {}
ExperimentConfig. The configuration of the experiments

global_config = {}
OrderedDict. The global configuration that applies to all projects

projects = {}
ProjectsConfig. The configuration of the projects

remove_experiment(*experiment*)

save()
Save the entire configuration files

class ExperimentsConfig(*projects*, *d=None*, *project_map=None*)
Bases: collections.OrderedDict

The configuration of the experiments

This class acts like a collections.OrderedDict but loads the experiment configuration only when you access the specific item (i.e. via d['exp_id'])

Parameters

- **projects** (*ProjectConfig*) – The project configuration
- **d** (*dict*) – An alternative dictionary to initialize from. If not given, the experiments are loaded on the fly from the *exp_files* attribute
- **project_map** (*dict*) – A mapping from project to experiments. If not given, it is created when accessing the *project_map* experiment

Methods

as_ordereddict()	Convenience method to convert this object into an OrderedDict
fix_paths(d[, root, project])	Fix the paths in the given dictionary to get absolute paths
items()	**Notes**
iteritems()	**Notes**
itervalues()	**Notes**
load()	Load all experiments in this dictionary into memory
rel_paths(d[, root, project])	Fix the paths in the given dictionary to get relative paths
remove(experiment)	Remove the configuration of an experiment
save()	Save the experiment configuration
values()	**Notes**

Attributes

exp_file	The path to the file containing all experiments in the configuration
exp_files	A mapping from experiment to experiment configuration file
paths	list of str. The keys describing paths for the model. Note that these
project_map	A mapping from project name to experiments

as_ordereddict()
Convenience method to convert this object into an OrderedDict

exp_file
The path to the file containing all experiments in the configuration

exp_files
A mapping from experiment to experiment configuration file

Note that this attribute only contains experiments whose configuration has already dumped to the file!

fix_paths(*d*, *root=None*, *project=None*)
Fix the paths in the given dictionary to get absolute paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project

- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

items() → a set-like object providing a view on D's items

Notes

Reimplemented to not load all experiments under python2.7

iteritems()

Notes

Reimplemented to not load all experiments under python2.7

itervalues()

Notes

Reimplemented to not load all experiments under python2.7

load()
Load all experiments in this dictionary into memory

paths = ['expdir', 'src', 'data', 'input', 'outdata', 'outdir', 'plot_output', 'projec
list of str. The keys describing paths for the model. Note that these keys here are replaced by the keys in the *paths* attribute of the specific *model_organization.ModelOrganizer* instance

project_map
A mapping from project name to experiments

rel_paths(*d*, *root=None*, *project=None*)
Fix the paths in the given dictionary to get relative paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project
- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

remove(*experiment*)
Remove the configuration of an experiment

save()
Save the experiment configuration

This method stores the configuration of each of the experiments in a file '<project-dir>/.project/<experiment>.yml', where '<project-dir>' corresponds to the project directory of the specific '<experiment>'. Furthermore it dumps all experiments to the *exp_file* configuration file.

values() → an object providing a view on D's values

Notes

Reimplemented to not load all experiments under python2.7

class ProjectsConfig(*conf_dir*, *d=None*)
Bases: collections.OrderedDict

The project configuration

This class stores the configuration from the projects, where each key corresponds to the name of one project and the value to the corresponding configuration.

Instances of this class are initialized by a file 'projects.yml' in the configuration directory (see the *all_projects* attribute) that stores a mapping from project name to project directory path. The configuration for each individual project is then loaded from the '<project-dir>/.project/.project.yml' file

Notes

Attributes

all_projects	The name of the configuration file
conf_dir	The path to the configuration directory
paths	list of str. The keys describing paths for the model. Note that these

Methods

fix_paths(d[, root, project])	Fix the paths in the given dictionary to get absolute paths
rel_paths(d[, root, project])	Fix the paths in the given dictionary to get relative paths
save()	Save the project configuration

If you move one project has been moved to another directory, make sure to update the 'projects.yml' file (the rest is updated when loading the configuration)

Parameters

- **conf_dir** (*str*) – The path to the configuration directory containing a file called `'projects.yml'`
- **d** (*dict*) – A dictionary to use to setup this configuration instead of loading them from the disk

all_projects
The name of the configuration file

conf_dir = None
The path to the configuration directory

fix_paths(*d*, *root=None*, *project=None*)
Fix the paths in the given dictionary to get absolute paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project
- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

paths = ['expdir', 'src', 'data', 'input', 'outdata', 'outdir', 'plot_output', 'projec
list of str. The keys describing paths for the model. Note that these keys here are replaced by the keys in the *paths* attribute of the specific *model_organization.ModelOrganizer* instance

rel_paths(*d*, *root=None*, *project=None*)
Fix the paths in the given dictionary to get relative paths

Parameters

- **d** (*dict*) – One experiment configuration dictionary
- **root** (*str*) – The root path of the project
- **project** (*str*) – The project name

Returns The modified *d*

Return type dict

Notes

d is modified in place!

save()
Save the project configuration

This method dumps the configuration for each project and the project paths (see the *all_projects* attribute) to the hard drive

get_configdir(*name*)
Return the string representing the configuration directory.

The directory is chosen as follows:

1. If the `name.upper() + CONFIGDIR` environment variable is supplied, choose that.

2a. On Linux, choose *$HOME/.config*.

2b. On other platforms, choose *$HOME/.matplotlib*.

3. If the chosen directory exists, use that as the configuration directory.

4. A directory: return None.

Notes

This function is taken from the matplotlib [1] module

References

[1]: http://matplotlib.org/api/

ordered_yaml_dump(*data*, *stream=None*, *Dumper=None*, ***kwds*)
Dumps the stream from an OrderedDict. Taken from

http://stackoverflow.com/questions/5121931/in-python-how-can-you-load-yaml- mappings-as-ordereddicts

ordered_yaml_load(*stream*, *Loader=None*, *object_pairs_hook=<class 'collections.OrderedDict'>*)
Loads the stream into an OrderedDict. Taken from

http://stackoverflow.com/questions/5121931/in-python-how-can-you-load-yaml- mappings-as-ordereddicts

safe_dump(*d*, *fname*, **args*, ***kwargs*)
Savely dump *d* to *fname* using yaml

This method creates a copy of *fname* called `fname + '~'` before saving *d* to *fname* using *ordered_yaml_dump()*

Parameters

- **d** (*object*) – The object to dump
- **fname** (*str*) – The path where to dump *d*

Other Parameters **‘‘*args, **kwargs‘‘** – Will be forwarded to the *ordered_yaml_dump()* function

safe_load(*fname*)
Load the file fname and make sure it can be done in parallel

Parameters **fname** (*str*) – The path name

setup_logging(*default_path=None*, *default_level=20*, *env_key=None*)
Setup logging configuration

Parameters

- **default_path** (*str*) – Default path of the yaml logging configuration file. If None, it defaults to the ‘logging.yaml’ file in the config directory
- **default_level** (*int*) – Default: `logging.INFO`. Default level if default_path does not exist
- **env_key** (*str*) – environment variable specifying a different logging file than *default_path* (Default: ‘LOG_CFG’)

Returns **path** – Path to the logging configuration file

Return type str

Notes

Function taken from http://victorlin.me/posts/2012/08/26/good-logging-practice-in-python

model_organization.utils module

Functions

dir_contains(dirname, path[, exists])	Check if a file of directory is contained in another.
get_module_path(mod)	Convenience method to get the directory of a given python module
get_next_name(old[, fmt])	Return the next name that numerically follows *old*
get_toplevel_module(mod)	
go_through_dict(key, d[, setdefault])	Split up the *key* by .
isstring(s)	
safe_list(l)	Function to create a list

dir_contains(*dirname*, *path*, *exists=True*)
Check if a file of directory is contained in another.

Parameters

- **dirname** (*str*) – The base directory that should contain *path*
- **path** (*str*) – The name of a directory or file that should be in *dirname*
- **exists** (*bool*) – If True, the *path* and *dirname* must exist

Notes

path and *dirname* must be either both absolute or both relative paths

get_module_path(*mod*)
Convenience method to get the directory of a given python module

get_next_name(*old*, *fmt='%i'*)
Return the next name that numerically follows *old*

get_toplevel_module(*mod*)

go_through_dict(*key*, *d*, *setdefault=None*)
Split up the *key* by . and get the value from the base dictionary *d*

Parameters

- **key** (*str*) – The key in the *config* configuration. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.
- **d** (*dict*) – The configuration dictionary containing the key
- **setdefault** (*callable*) – If not None and an item is not existent in *d*, it is created by calling the given function

Returns

- *str* – The last level of the key
- *dict* – The dictionary in *d* that contains the last level of the key

isstring(*s*)

safe_list(*l*)

Function to create a list

Parameters **l** (*iterable or anything else*) – Parameter that shall be converted to a list.

- If string or any non-iterable, it will be put into a list
- if iterable, it will be converted to a list

Returns *l* put (or converted) into a list

Return type list

1.4 Command Line API Reference

1.4.1 model setup

Perform the initial setup for the project

```
usage: model setup [-h] [-p str] [-link] str
```

Positional Arguments

str The path to the root directory where the experiments, etc. will be stored

Named Arguments

-p, --projectname The name of the project that shall be initialized at *root_dir*. A new directory will be created namely `root_dir + '/' + projectname`

-link If set, the source files are linked to the original ones instead of copied

Default: False

1.4.2 model init

Initialize a new experiment

```
usage: model init [-h] [-p str] [-d str]
```

Named Arguments

-p, --projectname The name of the project that shall be used. If None, the last one created will be used

-d, --description A short summary of the experiment

If the experiment is None, a new experiment will be created

If the experiment is None, a new experiment will be created

1.4.3 model set-value

Set a value in the configuration

```
usage: model set-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dt str]
                       level0.level1.level...=value
                       [level0.level1.level...=value ...]
```

Positional Arguments

level0.level1.level...=value The key-value pairs to set. If the configuration goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dt, --dtype Possible choices: ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupError, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, ZeroDivisionError, __build_class__, __debug__, __doc__, __import__, __loader__,

__name__, __package__, __spec__, abs, all, any, ascii, bin, bool, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset, getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip

The name of the data type or a data type to cast the value to

1.4.4 model get-value

Get one or more values in the configuration

```
usage: model get-value [-h] [-ep] [-pp] [-a] [-P] [-g] [-p str] [-nf] [-k]
                       [-b str] [-arc]
                       level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to get the values of. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

1.4.5 model del-value

Delete a value in the configuration

```
usage: model del-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dtype DTYPE]
                       level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to be deleted. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dtype

1.4.6 model info

Print information on the experiments

```
usage: model info [-h] [-ep] [-pp] [-gp] [-cp] [-a] [-nf] [-P] [-g] [-p str]
                  [-k] [-arc]
```

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-gp, --global-path If True/set, print the filename on the global configuration

Default: False

-cp, --config-path If True/set, print the path to the configuration directory

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

1.4.7 model unarchive

Extract archived experiments

```
usage: model unarchive [-h] [-ids exp1,[exp2[,...]]] [exp1,[exp2[,...]]] ...]]
                       [-f str] [-a] [-pd] [-P] [-d str] [-p str]
                       [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [--force]
```

Named Arguments

-ids, --experiments The experiments to extract. If None the current experiment is used

-f, --file The path to an archive to extract the experiments from. If None, we assume that the path to the archive has been stored in the configuration when using the `archive()` command

-a, --all If True, archives are extracted completely, not only the experiment (implies `project_data = True`)

Default: False

-pd, --project-data If True, the data for the project is extracted as well

Default: False

-P, --replace-project-config If True and the project does already exist in the configuration, it is updated with what is stored in the archive

Default: False

-d, --root An alternative root directory to use. Otherwise the experiment will be exctracted to

1. the root directory specified in the configuration files (if the project exists in it) and *replace_project_config* is False
2. the root directory as stored in the archive

-p, --projectname The projectname to use. If None, use the one specified in the archive

-fmt The format of the archive. If None, it is inferred

--force If True, force to overwrite the configuration of all experiments from what is stored in *archive*. Otherwise, the configuration of the experiments in *archive* are only used if missing in the current configuration

Default: False

1.4.8 model configure

Configure the project and experiments

```
usage: model configure [-h] [-g] [-p] [-i str] [-f str] [-s] [-n int or 'all']
                       [-update-from str]
```

Named Arguments

-g, --globally If True/set, the configuration are applied globally (already existing and configured experiments are not impacted)

Default: False

-p, --project Apply the configuration on the entire project instance instead of only the single experiment (already existing and configured experiments are not impacted)

Default: False

-i, --ifile The input file for the project. Must be a netCDF file with population data

-f, --forcing The input file for the project containing variables with population evolution information. Possible variables in the netCDF file are *movement* containing the number of people to move and *change* containing the population change (positive or negative)

-s, --serial Do the parameterization always serial (i.e. not in parallel on multiple processors). Does automatically impact global settings

Default: False

-n, --nprocs Maximum number of processes to when making the parameterization in parallel. Does automatically impact global settings and disables *serial*

-update-from Path to a yaml configuration file to update the specified configuration with it

1.4.9 model archive

Archive one or more experiments or a project instance

This method may be used to archive experiments in order to minimize the amount of necessary configuration files

```
usage: model archive [-h] [-d str] [-f str]
                     [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [-p str]
                     [-ids exp1,[exp2[,...]]]] [-P] [-na] [-np]
                     [-e str [str ...]] [-k] [-rm] [-n] [-L]
```

Named Arguments

-d, --odir The path where to store the archive

-f, --aname The name of the archive (minus any format-specific extension). If None, defaults to the projectname

-fmt Possible choices: bztar, gztar, tar, zip

The format of the archive. If None, it is tested whether an archived with the name specified by *aname* already exists and if yes, the format is inferred, otherwise `'tar'` is used

-p, --projectname If provided, the entire project is archived

-ids, --experiments If provided, the given experiments are archived. Note that an error is raised if they belong to multiple project instances

-P, --current-project If True, *projectname* is set to the current project

Default: False

-na, --no-append It True and the archive already exists, it is deleted

Default: False

-np, --no-project-paths If True, paths outside the experiment directories are neglected

Default: False

-e, --exclude Filename patterns to ignore (see `glob.fnmatch.fnmatch()`)

-k, --keep If True, the experiment directories are not removed and no modification is made in the configuration

Default: False

-rm, --rm-project If True, remove all the project files

Default: False

-n, --dry-run If True, set, do not actually make anything

Default: False

-L, --dereference If set, dereference symbolic links. Note: This is automatically set for `fmt=='zip'`

Default: False

1.4.10 model remove

Delete an existing experiment and/or projectname

```
usage: model remove [-h] [-p [str]] [-a] [-y] [-ap]
```

Named Arguments

-p, --projectname The name for which the data shall be removed. If set without, argument, the project will be determined by the experiment. If specified, all experiments for the given project will be removed.

-a, --all If set, delete not only the experiments and config files, but also all the project files

Default: False

-y, --yes If True/set, do not ask for confirmation

Default: False

-ap, --all-projects If True/set, all projects are removed

Default: False

The main function for parsing global arguments

```
usage: model [-h] [-id str] [-l] [-n] [-v] [-vl str or int] [-nm] [-E]
             {setup,init,set-value,get-value,del-value,info,unarchive,configure,
↪archive,remove}
             ...
```

1.4.11 Named Arguments

-id, --experiment **experiment: str** The id of the experiment to use. If the *init* argument is called, the *new* argument is automatically set. Otherwise, if not specified differently, the last created experiment is used.

-l, --last If True, the last experiment is used

Default: False

-n, --new If True, a new experiment is created

Default: False

-v, --verbose Increase the verbosity level to DEBUG. See also *verbosity_level* for a more specific determination of the verbosity

Default: False

-vl, --verbosity-level The verbosity level to use. Either one of `'DEBUG'`, `'INFO'`, `'WARNING'`, `'ERROR'` or the corresponding integer (see pythons logging module)

-nm, --no-modification If True/set, no modifications in the configuration files will be done

Default: False

-E, --match If True/set, interprete *experiment* as a regular expression (regex) und use the matching experiment

Default: False

1.4.12 Sub-commands:

setup

Perform the initial setup for the project

```
model setup [-h] [-p str] [-link] str
```

Positional Arguments

str The path to the root directory where the experiments, etc. will be stored

Named Arguments

-p, --projectname The name of the project that shall be initialized at *root_dir*. A new directory will be created namely `root_dir + '/' + projectname`

-link If set, the source files are linked to the original ones instead of copied

Default: False

init

Initialize a new experiment

```
model init [-h] [-p str] [-d str]
```

Named Arguments

-p, --projectname The name of the project that shall be used. If None, the last one created will be used

-d, --description A short summary of the experiment

Notes

If the experiment is None, a new experiment will be created

Notes

If the experiment is None, a new experiment will be created

set-value

Set a value in the configuration

```
model set-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dt str]
                level0.level1.level...=value
                [level0.level1.level...=value ...]
```

Positional Arguments

level0.level1.level...=value The key-value pairs to set. If the configuration goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dt, --dtype Possible choices: ArithmeticError, AssertionError, AttributeError, BaseException, BlockingIOError, BrokenPipeError, BufferError, BytesWarning, ChildProcessError, ConnectionAbortedError, ConnectionError, ConnectionRefusedError, ConnectionResetError, DeprecationWarning, EOFError, Ellipsis, EnvironmentError, Exception, False, FileExistsError, FileNotFoundError, FloatingPointError, FutureWarning, GeneratorExit, IOError, ImportError, ImportWarning, IndentationError, IndexError, InterruptedError, IsADirectoryError, KeyError, KeyboardInterrupt, LookupError, MemoryError, ModuleNotFoundError, NameError, None, NotADirectoryError, NotImplemented, NotImplementedError, OSError, OverflowError, PendingDeprecationWarning, PermissionError, ProcessLookupError, RecursionError, ReferenceError, ResourceWarning, RuntimeError, RuntimeWarning, StopAsyncIteration, StopIteration, SyntaxError, SyntaxWarning, SystemError, SystemExit, TabError, TimeoutError, True, TypeError, UnboundLocalError, UnicodeDecodeError, UnicodeEncodeError, UnicodeError, UnicodeTranslateError, UnicodeWarning, UserWarning, ValueError, Warning, ZeroDivisionError, __build_class__, __debug__, __doc__, __import__, __loader__, __name__, __package__, __spec__, abs, all, any, ascii, bin, bool, bytearray, bytes, callable, chr, classmethod, compile, complex, copyright, credits, delattr, dict, dir, divmod, enumerate, eval, exec, exit, filter, float, format, frozenset,

getattr, globals, hasattr, hash, help, hex, id, input, int, isinstance, issubclass, iter, len, license, list, locals, map, max, memoryview, min, next, object, oct, open, ord, pow, print, property, quit, range, repr, reversed, round, set, setattr, slice, sorted, staticmethod, str, sum, super, tuple, type, vars, zip

The name of the data type or a data type to cast the value to

get-value

Get one or more values in the configuration

```
model get-value [-h] [-ep] [-pp] [-a] [-P] [-g] [-p str] [-nf] [-k] [-b str]
                [-arc]
                level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to get the values of. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `'.'`, it must be escaped by a preceeding `''`.

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

del-value

Delete a value in the configuration

```
model del-value [-h] [-a] [-P] [-g] [-p str] [-b str] [-dtype DTYPE]
                level0.level1.level... [level0.level1.level... ...]
```

Positional Arguments

level0.level1.level... A list of keys to be deleted. If the key goes some levels deeper, keys may be separated by a `'.'` (e.g. `'namelists.weathergen'`). Hence, to insert a `','`, it must be escaped by a preceeding `''`.

Named Arguments

-a, --all If True/set, the information on all experiments are printed

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-b, --base A base string that shall be put in front of each key in *values* to avoid typing it all the time

Default: “”

-dtype

info

Print information on the experiments

```
model info [-h] [-ep] [-pp] [-gp] [-cp] [-a] [-nf] [-P] [-g] [-p str] [-k]
           [-arc]
```

Named Arguments

-ep, --exp-path If True/set, print the filename of the experiment configuration

Default: False

-pp, --project-path If True/set, print the filename on the project configuration

Default: False

-gp, --global-path If True/set, print the filename on the global configuration

Default: False

-cp, --config-path If True/set, print the path to the configuration directory

Default: False

-a, --all If True/set, the information on all experiments are printed

Default: False

-nf, --no-fix If set, paths are given relative to the root directory of the project

Default: False

-P, --on-projects If set, show information on the projects rather than the experiment

Default: False

-g, --globally If set, show the global configuration settings

Default: False

-p, --projectname The name of the project that shall be used. If provided and *on_projects* is not True, the information on all experiments for this project will be shown

-k, --only-keys If True, only the keys of the given dictionary are printed

Default: False

-arc, --archives If True, print the archives and the corresponding experiments for the specified project

Default: False

unarchive

Extract archived experiments

```
model unarchive [-h] [-ids exp1,[exp2[,...]]] [exp1,[exp2[,...]]] ...]]
                [-f str] [-a] [-pd] [-P] [-d str] [-p str]
                [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [--force]
```

Named Arguments

-ids, --experiments The experiments to extract. If None the current experiment is used

-f, --file The path to an archive to extract the experiments from. If None, we assume that the path to the archive has been stored in the configuration when using the `archive()` command

-a, --all If True, archives are extracted completely, not only the experiment (implies `project_data = True`)

Default: False

-pd, --project-data If True, the data for the project is extracted as well

Default: False

-P, --replace-project-config If True and the project does already exist in the configuration, it is updated with what is stored in the archive

Default: False

-d, --root An alternative root directory to use. Otherwise the experiment will be exctracted to

1. the root directory specified in the configuration files (if the project exists in it) and *replace_project_config* is False
2. the root directory as stored in the archive

-p, --projectname The projectname to use. If None, use the one specified in the archive

-fmt The format of the archive. If None, it is inferred

--force If True, force to overwrite the configuration of all experiments from what is stored in *archive*. Otherwise, the configuration of the experiments in *archive* are only used if missing in the current configuration

Default: False

configure

Configure the project and experiments

```
model configure [-h] [-g] [-p] [-i str] [-f str] [-s] [-n int or 'all']
                [-update-from str]
```

Named Arguments

-g, --globally If True/set, the configuration are applied globally (already existing and configured experiments are not impacted)

Default: False

-p, --project Apply the configuration on the entire project instance instead of only the single experiment (already existing and configured experiments are not impacted)

Default: False

-i, --ifile The input file for the project. Must be a netCDF file with population data

-f, --forcing The input file for the project containing variables with population evolution information. Possible variables in the netCDF file are *movement* containing the number of people to move and *change* containing the population change (positive or negative)

-s, --serial Do the parameterization always serial (i.e. not in parallel on multiple processors). Does automatically impact global settings

Default: False

-n, --nprocs Maximum number of processes to when making the parameterization in parallel. Does automatically impact global settings and disables *serial*

-update-from Path to a yaml configuration file to update the specified configuration with it

archive

Archive one or more experiments or a project instance

This method may be used to archive experiments in order to minimize the amount of necessary configuration files

```
model archive [-h] [-d str] [-f str]
              [-fmt { 'gztar' | 'bztar' | 'tar' | 'zip' }] [-p str]
              [-ids exp1,[exp2[,...]]]] [-P] [-na] [-np] [-e str [str ...]]
              [-k] [-rm] [-n] [-L]
```

Named Arguments

-d, --odir The path where to store the archive

-f, --aname The name of the archive (minus any format-specific extension). If None, defaults to the projectname

-fmt Possible choices: bztar, gztar, tar, zip

The format of the archive. If None, it is tested whether an archived with the name specified by *aname* already exists and if yes, the format is inferred, otherwise `'tar'` is used

-p, --projectname If provided, the entire project is archived

-ids, --experiments If provided, the given experiments are archived. Note that an error is raised if they belong to multiple project instances

-P, --current-project If True, *projectname* is set to the current project

Default: False

-na, --no-append It True and the archive already exists, it is deleted

Default: False

-np, --no-project-paths If True, paths outside the experiment directories are neglected

Default: False

-e, --exclude Filename patterns to ignore (see `glob.fnmatch.fnmatch()`)

-k, --keep If True, the experiment directories are not removed and no modification is made in the configuration

Default: False

-rm, --rm-project If True, remove all the project files

Default: False

-n, --dry-run If True, set, do not actually make anything

Default: False

-L, --dereference If set, dereference symbolic links. Note: This is automatically set for `fmt=='zip'`

Default: False

remove

Delete an existing experiment and/or projectname

```
model remove [-h] [-p [str]] [-a] [-y] [-ap]
```

Named Arguments

-p, --projectname	The name for which the data shall be removed. If set without, argument, the project will be determined by the experiment. If specified, all experiments for the given project will be removed.
-a, --all	If set, delete not only the experiments and config files, but also all the project files Default: False
-y, --yes	If True/set, do not ask for confirmation Default: False
-ap, --all-projects	If True/set, all projects are removed Default: False

1.5 Changelog

1.5.1 v0.1.10

Compatibility fix for python3.7

Changed

- A bug has been fixed for the archiving using pythons tarfile library for python 3.7

1.5.2 v0.1.9

Changed

- The `remove` command now removes the configuration files for the experiments, too

1.5.3 v0.1.8

Added

- LICENSE file

1.5.4 v0.1.7

Changed

- Minor bug fix in remove command

1.5.5 v0.1.6

Changed

- You can modify the behaviour of the logging via the environment variable `'LOG_' + ModelOrganizer.name.upper()` (previously, it was `ModelOrganizer.name.capitalize()`)
- The ExperimentsConfig class only loads the configuration when it is directly accessed via the `__getitem__` method, i.e. via `organizer.config.experiments[exp]`. Conversion to a dictionary, or the `items`, `values`, `iteritems`, and `itervalues` methods will not load the experiment

1.5.6 v0.1.5

Added

- Added as_ordereddict method for ExperimentsConfig that avoids a loading of all experiments when converting to an OrderedDict

1.5.7 v0.1.4

Changed

- Fixed bug that loads all experiments in Python2.7 when initializing a ExperimentsConfig

1.5.8 v0.1.3

Added

- Added fix_paths and rel_paths method for ProjectConfig to store only the relative paths

Changed

- Dump a deepcopy of the configuration when saving the experiment and project
- load all experiments in the info method if the *complete* parameter (`'-a'` flag) is True

1.5.9 v0.1.2

Added

- Added changelog

Changed

- The model_organization.config.setup_logging function is now called every new initialzation of a model_organizer if no config is provided

1.6 Installation

Simply install it via `pip`:

```
$ pip install model-organization
```

Or you install it via:

```
$ python setup.py install
```

from the source on GitHub.

1.7 Requirements

The package is based upon

- funcargparse: used to create the command line utility.
- PyYAML: for storing, loading and displaying the configuration
- six: For compatibility issues between python 2 and python 3
- fasteners: For a parallel access to the configuration files

The package is regularly tested for python 2.7, 3.4, 3.5 and 3.6.

1.8 Indices and tables

- genindex
- modindex
- search

Python Module Index

m

Index

O

P

R

S

T

U

V