MLOQ

Release 0.0.75

Guillem Duran, Vadim Markovtsev

Oct 16, 2022

WELCOME TO MLOQ

ML Ops Quickstart is a tool for initializing Machine Learning projects following ML Ops best practices.

Setting up new repositories is a time-consuming task that involves creating different files and configuring tools such as linters, docker containers and continuous integration pipelines. The goal of `mloq` is to simplify that process, so you can start writing code as fast as possible.

`mloq` generates customized templates for Python projects with focus on Maching Learning. An example of the generated templates can be found in mloq-template.

CHAPTER

ONE

ABOUT ML OPS QUICKSTART

ML Ops Quickstart is a tool for initializing Machine Learning projects following ML Ops best practices.

Setting up new repositories is a time-consuming task that involves creating different files and configuring tools such as linters, docker containers, and continuous integration pipelines. The goal of `mloq` is to simplify that process, so you can start writing code as fast as possible.

`mloq` generates customized templates for Python projects with a focus on Maching Learning. An example of the generated templates can be found in mloq-template.

CHAPTER

TWO

INSTALLATION

`mloq` is tested on Ubuntu 18.04+, and supports Python 3.6+.

2.1 Install from pypi

```
pip install mloq
```

2.2 Install from source

```
git clone https://github.com/FragileTech/ml-ops-quickstart.git
cd ml-ops-quickstart
pip install -e .
```

CHAPTER

THREE

COMMAND LINE INTERFACE

Options:

- `--file -f`: Name of the configuration file. If `file` is a directory, it will load the `mloq.yml` file present in it.
- `--overwrite -o`: Rewrite files that already exist in the target project.
- `--interactive -i`: Missing configuration data can be defined interactively from the CLI.

CHAPTER

FOUR

USAGE EXAMPLES

Arguments:

- `OUTPUT_DIRECTORY`: Path to the target project.

To set up a new repository from scratch interactively in the current working directory:

```
mloq setup -i .
```

To load a `mloq.yml` configuration file from the current repository, and initialize the directory `example`, and overwrite all existing files with no interactivity:

```
mloq setup -f . -o example
```

```
Welcome to the MLOQ 0.0.16 interactive setup utility.

Please enter values for the following settings (just press Enter to accept a default value, if one is given in brackets).

The following values will occur in several places in the generated files.
> Select project name: my-new-project
> Short description of the project: This is a short description
> Github handle of the project owner: owner
> Owner contact email: owner@mail.com
> Author(s) of the project [owner]: Owner and Employee
> GitHub project url [https://github.com/owner/my-new-project]:

Please define the license of the project.
Open Source projects will include the corresponding LICENSE file and a DCO.md file
> Is the project Open Source? [Y/n]:
> Project license type (MIT, None) [MIT]:
> Copyright holder [owner]: Owner Name

What Python versions are supported? Please provide the values as a comma separated list.
> Supported python versions [3.6, 3.7, 3.8, 3.9]:
Please specify the requirements of the project as a comma separated list.
Available values:
    data-science: Common data science libraries such as numpy, pandas, sklearn...
    data-viz: Visualization libraries such as holoviews, bokeh, plotly, matplotlib...
    pytorch: Latest version of pytorch, torchvision and pytorch_lightning
    tensorflow:
> Project requirements [None]: pytorch

You can configure the continuous integration using Github Actions.
Available values:
    Python: Push workflow for pure Python projects.
    Dist: Push workflow for Python projects with compiled extensions.
    None: Do not set up the CI.
> Github Actions push workflow (python, dist, none) [python]:
> Default branch of the project [master]: main
A bot account will be used to automatically bump the version of your project.
> Bot's GitHub login to push commits in CI [Owner and Employee]: company-bot
> Bot account email [owner@mail.com]: bot@company.com
MLOQ will generate a Dockerfile for your project.
> Base docker image for the project's Docker container [fragiletech/ubuntu18.04-cuda-11.0-py39]:
You can optionally create an ML Flow MLproject file.
> Do you want to set up ML Flow? [y/N]:
Do you want to generate a mloq.yml file? [y/N]: y
Do you want to override existing files? [y/N]:
```

CHAPTER

FIVE

MLOQ.YML CONFIG FILE

This yaml file contains all the information used by mloq to set up a new project. All values are strings except **python_versions** and **requirements**, which are lists of strings. **null** values are interpreted as missing values.

```yaml
# This yaml file contains all the information used by mloq to set up a new project.
# All values in template are strings and booleans,
# except "python_versions" and "requirements" that are lists of strings.
# "null" values are interpreted as non-defined values.
# ----------------------------------------------------------------------------

# project_config values are necessary to define the files that will be written, and the tools
# that will be configured.
project_config:
  open_source: null  # boolean. If True, set up and Open Source project
  docker: null  # boolean If True, set up a Docker image for the project
  ci: null  # Name of the GitHub Actions CI workflow that will be configured.
  mlflow: null # boolean. If True configure a MLproject file compatible with ML Flow projects.
  requirements: null # List containing the pre-defined requirements of the project.

# template contains all the values that will be written in the generated files.
# They are loaded as a dictionary and passed to jinja2 to fill in the templates.
template:
  project_name: null  # Name of the new Python project
  default_branch: null  # Name of the default branch. Used in the CI push workflow.
  owner: null  # Github handle of the project owner
  author: null  # Person(s) or entity listed as the project author in setup.py
  email: null  # Owner contact email
  copyright_holder: null  # Owner of the project copyright.
  project_url: null  # Project download url. Defaults to https://github.com/{owner}/{project_name}
  bot_name: null  # GitHub login of the account used to push when bumping the project version
  bot_email: null # Bot account email
  license: null  # Currently only proprietary and MIT license is supported
  description: null  # Short description of the project
  python_versions: null # Supported Python versions
  docker_image: null  # Your project Docker container will inherit from this image.
```

`mloq` offers a set of features focused on generating automatically all the required files to start your Machine Learning project.

CHAPTER

SIX

REPOSITORY FILES

Set up the following common repository files personalized for your project with the values defined in `mloq.yml`:

- README.md
- DCO.md
- CONTRIBUTING.md
- CODE_OF_CONDUCT.md
- LICENSE
- .gitignore

CHAPTER

SEVEN

PACKAGING

Automatic configuration of `pyproject.toml` and `setup.py` to distribute your project as a Python package.

CHAPTER

EIGHT

CODE STYLE

All the necessary configuration for the following tools is defined in pyproject.toml.

- black: Automatic code formatter.
- isort: Rearrange your imports automatically.
- flakehell: Linter tool build on top of `flake8`, `pylint` and `pycodestyle`

CHAPTER

NINE

REQUIREMENTS

`mloq` creates three different requirements files in the root directory of the project. Each file contains pinned dependencies.

- requirements-lint.txt: Contains the dependencies for running style check analysis and automatic formatting of the code.
- requirements-test.txt:

Dependencies for running pytest, hypothesis, and test coverage.

- `requirements.txt`: Contains different pre-configured dependencies that can be defined in `mloq.yml`. The available pre-configured dependencies are:
  - data-science: Dependencies of common data science libraries.
  - data-visualization: Common visualization libraries.
  - Last version of pytorch and tensorflow

CHAPTER

TEN

DOCKER

A Dockerfile that builds a container on top of the FragileTech Docker Hub images:

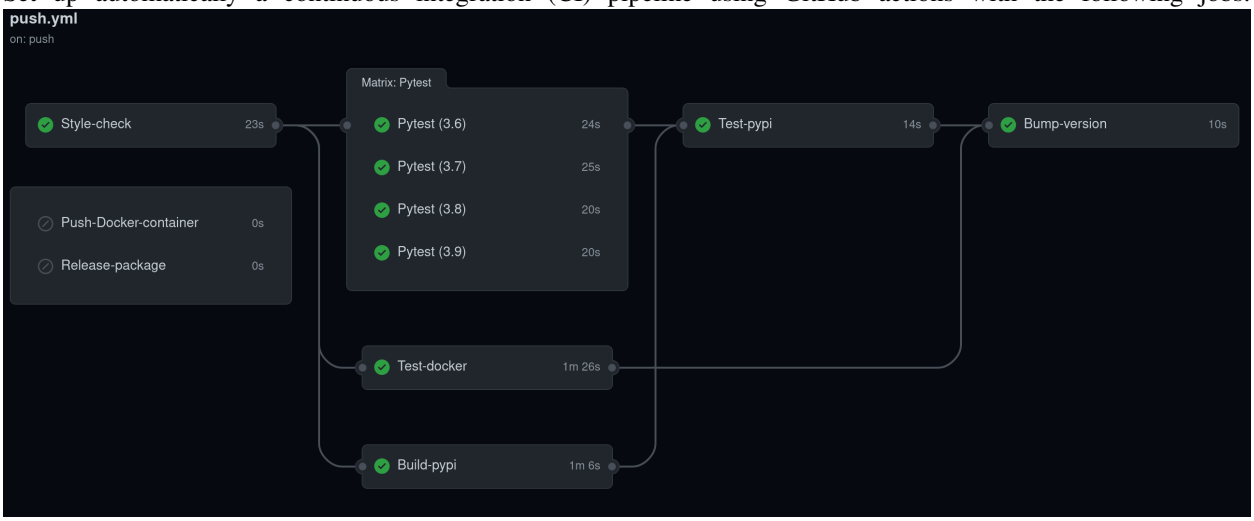
- If *tensorflow* or *pytorch* are selected as requirements, the container has CUDA 11.0 installed.
- Installs all the packages listed in `requirements.txt`.
- Installs `requirements-test.txt` and `requirements-lint.txt` dependencies.
- Install a `jupyter notebook` server with a configurable password on port 8080.
- Installs the project with `pip install -e .`.

CHAPTER

ELEVEN

CONTINUOUS INTEGRATION USING GITHUB ACTIONS

Set up automatically a continuous integration (CI) pipeline using GitHub actions with the following jobs:



Automatic build and tests:

- **Style Check**: Run `flake8` and `black --check` to ensure a consistent code style.
- **Pytest**: Test the project using pytest on all supported Python versions and output a code coverage report.
- **Test-docker**: Build the project's Docker container and run the tests inside it.
- **Build-pypi**: Build the project and upload it to Test Pypi with a version tag unique to each commit.
- **Test-pypi**: Install the project from Test Pypi and run the tests using pytest.
- **Bump-version**: Automatically bump the project version and create a tag in the repository every time the default branch is updated.

Deploy each new version:

- **Push-docker-container**: Upload the project's Docker container to Docker Hub.
- **Release-package**: Upload to Pypi the source of the project and the corresponding wheels.

CHAPTER

TWELVE

TESTING

The last versions of `pytest`, `hypothesis`, and `pytest-cov` can be found in `requirements-test.txt`.

The folder structure for the library and tests is created.

CHAPTER

THIRTEEN

PROJECT MAKEFILE

A `Makefile` will be created in the root directory of the project. It contains the following commands:

- `make style`: Run `isort` and `black` to automatically arrange the imports and format the project.
- `make check`: Run `flakehell` and check black style. If it raises any error the CI will fail.
- `make test`: Clear the tests cache and run pytest.
- `make pipenv-install`: Install the project in a new Pipenv environment and create a new `Pipfile` and `Pipfile.lock`.
- `make pipenv-test`: Run pytest inside the project's Pipenv.
- `make docker-build`: Build the project's Docker container.
- `make docker-test`: Run pytest inside the projects docker container.
- `make docker-shell`: Mount the current project as a docker volume and open a terminal in the project's container.
- `make docker-notebook`: Mount the current project as a docker volume and open a jupyter notebook in the project's container. It exposes the notebook server on the port `8080`.

CHAPTER

FOURTEEN

LICENSE

ML Ops Quickstart is released under the MIT license.

CHAPTER

FIFTEEN

CONTRIBUTING

Contributions are very welcome! Please check the contributing guidelines before opening a pull request.

CHAPTER

SIXTEEN

CONTRIBUTING GUIDELINES

mloq is *MIT licensed* and accepts contributions via GitHub pull requests. This document outlines some of the conventions on development workflow, commit message formatting, contact points, and other resources to make it easier to get your contribution accepted.

16.1 Certificate of Origin

By contributing to this project you agree to the Developer Certificate of Origin (DCO). This document was created by the Linux Kernel community and is a simple statement that you, as a contributor, have the legal right to make the contribution.

In order to show your agreement with the DCO you should include at the end of commit message, the following line: `Signed-off-by: John Doe <john.doe@example.com>`, using your real name.

This can be done easily using the `-s` flag on the `git commit`.

16.2 Support Channels

The official support channels, for both users and contributors, are:

- GitHub issues*

*Before opening a new issue or submitting a new pull request, it's helpful to search the project - it's likely that another user has already reported the issue you're facing, or it's a known issue that we're already aware of.

16.3 How to Contribute

Pull Requests (PRs) are the main and exclusive way to contribute to the official project. In order for a PR to be accepted it needs to pass a list of requirements:

- The CI style check passes (run locally with `make check`).
- Code Coverage does not decrease.
- All the tests pass.
- Python code is formatted according to
- If the PR is a bug fix, it has to include a new unit test that fails before the patch is merged.
- If the PR is a new feature, it has to come with a suite of unit tests, that tests the new functionality.

- In any case, all the PRs have to pass the personal evaluation of at least one of the maintainers.

16.3.1 Format of the commit message

The commit summary must start with a capital letter and with a verb in present tense. No dot in the end.

```
Add a feature
Remove unused code
Fix a bug
```

Every commit details should describe what was changed, under which context and, if applicable, the GitHub issue it relates to.

16.4 Roadmap

- [ ] Improve documentation and test coverage.
- [ ] Configure `sphinx` to build the docs automatically.
- [ ] Implement checks for additional best practices.
- [ ] Improve command-line interface and logging.
- [ ] Add new customization options.

CHAPTER

SEVENTEEN

API REFERENCE

This page holds mloq API documentation, which might be helpful for final users or developers to create their own mloq-based utilities. Among the different sub-packages and modules, we might differentiate two big categories: core utilities and high-level ones.

- **Core API:** This routines are located within the *mloq.api* that show the different features of the library can be accessed programmatically.

17.1 mloq

Package for initializing ML projects following ML Ops best practices.

17.1.1 Subpackages

17.1.1.1 mloq.commands

Contains all the defined mloq Commands.

17.1.1.1.1 Submodules

mloq.commands.ci

Mloq ci command implementation.

Module Contents

Classes

CiCMD	Implement the functionality of the ci Command.

Attributes

CI_ASSETS_PATH
push_python_wkf
WORKFLOW_FILES

mloq.commands.ci.**CI_ASSETS_PATH**

mloq.commands.ci.**push_python_wkf**

mloq.commands.ci.**WORKFLOW_FILES**

class mloq.commands.ci.**CiCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the ci Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name :str = ci

ubuntu_version

disable

docker

project_name

default_branch

docker_org

bot_name

bot_email

ci_python_version

python_versions

ci_extra

vendor

open_source

author

owner

email

project_url

files

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.docker

Mloq docker command implementation.

Module Contents

Classes

DockerCMD	Implement the functionality of the docker Command.

Attributes

DOCKER_ASSETS_PATH
dockerfile
makefile_docker
DOCKER_FILES

mloq.commands.docker.**DOCKER_ASSETS_PATH**

mloq.commands.docker.**dockerfile**

mloq.commands.docker.**makefile_docker**

mloq.commands.docker.**DOCKER_FILES**

class mloq.commands.docker.**DockerCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the docker Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = docker

files

disable

cuda

cuda_image_type

cuda_version

ubuntu_version

project_name

docker_org

python_version

base_image

test

lint

jupyter

jupyter_password

requirements

extra

makefile

static require_cuda_from_requirements(*project_config=None*)

Return True if any of the project dependencies require CUDA.

Parameters **project_config** (*Optional[omegaconf.DictConfig]*) –

Return type bool

requires_cuda()

Return True if the Docker container requires CUDA.

Return type bool

get_base_image()

Return the name of the base image for the project Docker container.

parse_config()

Update the configuration dictionary from the data entered by the user.

Return type omegaconf.DictConfig

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.docs

Mloq docs command implementation.

Module Contents

Classes

DocsCMD	Implement the functionality of the docs Command.

Attributes

DOCS_ASSETS_PATH
makefile_docs
make_bat_docs
docs_req
conf_py
index_md
deploy_docs
DOCS_FILES

mloq.commands.docs.**DOCS_ASSETS_PATH**

mloq.commands.docs.**makefile_docs**

mloq.commands.docs.**make_bat_docs**

mloq.commands.docs.**docs_req**

mloq.commands.docs.**conf_py**

mloq.commands.docs.**index_md**

mloq.commands.docs.**deploy_docs**

mloq.commands.docs.**DOCS_FILES**

class mloq.commands.docs.**DocsCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the docs Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = docs

disable

project_name

description

author

copyright_year

copyright_holder

deploy_docs

default_branch

project_url

files

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.globals

Mloq globals command implementation.

Module Contents

Classes

GlobalsCMD	Implement the functionality of the globals Command.

class mloq.commands.globals.**GlobalsCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the globals Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = globals

project_name

description

author

owner

email

open_source

project_url

default_branch

license

use_poetry

main_python_version

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

parse_config()

Generate the configuration of the project via a configuration file.

Return type omegaconf.DictConfig

mloq.commands.license

Mloq license command implementation.

Module Contents

Classes

LicenseCMD	Implement the functionality of the license Command.

Attributes

TEMPLATES_PATH
dco
mit_license
apache_license
gpl_license
LICENSES
LICENSE_FILES

mloq.commands.license.**TEMPLATES_PATH**

mloq.commands.license.**dco**

mloq.commands.license.**mit_license**

mloq.commands.license.**apache_license**

mloq.commands.license.**gpl_license**

mloq.commands.license.**LICENSES**

mloq.commands.license.**LICENSE_FILES**

class mloq.commands.license.**LicenseCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the license Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = license

files

LICENSES

disable

license

copyright_year

copyright_holder

project_name

project_url

email

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.lint

Mloq lint command implementation.

Module Contents

Classes

LintCMD	Implement the functionality of the lint Command.

Attributes

lint_req
LINT_FILES

mloq.commands.lint.**lint_req**

mloq.commands.lint.**LINT_FILES**

class mloq.commands.lint.**LintCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the lint Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = lint

files

disable

black

isort

linters

docstring_checks

pyproject_extra

project_name

makefile

poetry_requirements

ignore_files

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.package

Mloq package command implementation.

Module Contents

Classes

PackageCMD	Implement the functionality of the package Command.

Attributes

PACKAGE_ASSETS_PATH
PYTHON_VERSIONS
DEFAULT_PYTHON_VERSIONS
setup_py
PACKAGE_FILES

mloq.commands.package.**PACKAGE_ASSETS_PATH**

mloq.commands.package.**PYTHON_VERSIONS = ['3.6', '3.7', '3.8', '3.9', '3.10']**

mloq.commands.package.**DEFAULT_PYTHON_VERSIONS = ['3.7', '3.8', '3.9', '3.10']**

mloq.commands.package.**setup_py**

mloq.commands.package.**PACKAGE_FILES**

class mloq.commands.package.**PackageCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the package Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = package

files

LICENSE_CLASSIFIERS

disable

pyproject_extra

project_name

license

license_classifier

description

default_branch

project_url

owner

author

email

main_python_version

python_versions

use_poetry

parse_config()

Update the configuration DictConfig with the Command parameters.

Return type omegaconf.DictConfig

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.project

Mloq project command implementation.

Module Contents

Classes

ProjectCMD	Implement the functionality of the project Command.

Attributes

PROJECT_ASSETS_PATH
readme
gitignore
pre_commit_hook
init
main
test_main
test_req
version
code_of_conduct
codecov
contributing
PROJECT_FILES

mloq.commands.project.**PROJECT_ASSETS_PATH**

mloq.commands.project.**readme**

mloq.commands.project.**gitignore**

mloq.commands.project.**pre_commit_hook**

mloq.commands.project.**init**

mloq.commands.project.**main**

mloq.commands.project.**test_main**

mloq.commands.project.**test_req**

mloq.commands.project.**version**

mloq.commands.project.**code_of_conduct**

mloq.commands.project.**codecov**

mloq.commands.project.**contributing**

mloq.commands.project.**PROJECT_FILES**

class mloq.commands.project.**ProjectCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the project Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = project

files

disable

project_name

owner

description

project_url

license

tests

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.requirements

Mloq requirements command implementation.

Module Contents

Classes

RequirementsCMD	Implement the functionality of the requirements Command.

Attributes

REQUIREMENTS_PATH
requirements
data_science_req
data_viz_req
pytorch_req
tensorflow_req
lint_req
test_req
dogfood_req
docs_req
REQUIREMENTS_FILES
REQUIREMENT_CHOICES

mloq.commands.requirements.**REQUIREMENTS_PATH**

mloq.commands.requirements.**requirements**

mloq.commands.requirements.**data_science_req**

mloq.commands.requirements.**data_viz_req**

mloq.commands.requirements.**pytorch_req**

mloq.commands.requirements.**tensorflow_req**

mloq.commands.requirements.**lint_req**

mloq.commands.requirements.**test_req**

mloq.commands.requirements.**dogfood_req**

mloq.commands.requirements.**docs_req**

mloq.commands.requirements.**REQUIREMENTS_FILES**

mloq.commands.requirements.**REQUIREMENT_CHOICES = ['data-science', 'data-viz', 'torch', 'tensorflow', 'none', 'dogfood', 'None']**

class mloq.commands.requirements.**RequirementsCMD**(*record*, *interactive=False*)

Bases: *mloq.command.Command*

Implement the functionality of the requirements Command.

Parameters

- **record** (mloq.record.CMDRecord) –
- **interactive** (*bool*) –

cmd_name = requirements

files

disable

requirements

REQUIREMENTS_ALIASES

__del__()

Remove the temporary directory when the instance is deleted.

Return type None

classmethod get_aliased_requirements_file(*option*)

Get requirement file from aliased name.

Parameters **option** (*str*) –

Return type *mloq.files.File*

classmethod read_requirements_file(*option*)

Return the content of the target requirements file form an aliased name.

Parameters **option** (*str*) –

Return type str

classmethod compose_requirements(*options*)

Return the content requirements.txt file with pinned dependencies.

The returned string contains the combined dependencies for the different options sorted alphabetically.

Parameters **options** (*Iterable[str]*) – Iterable containing the aliased names of the target dependencies for the project.

Returns str containing the pinned versions of all the selected requirements.

Return type str

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

static requirements_is_empty(*options*)

Return True if no requirements are specified for the project.

Parameters **options** (*Union[List[str], str]*) –

Return type bool

record_files()

Register the files that will be generated by mloq.

Return type None

mloq.commands.setup

Mloq setup command implementation.

Module Contents

Classes

SetupCMD	Implement the functionality of the setup Command.

Functions

_sub_commands()

Attributes

SUB_COMMANDS

mloq.commands.setup.**_sub_commands**()

mloq.commands.setup.**SUB_COMMANDS**

class mloq.commands.setup.**SetupCMD**(*record*, *interactive=False*)

Bases: *mloq.command.CommandMixin*

Implement the functionality of the setup Command.

Parameters

- **record** (mloq.record.CMDRecord) –
- **interactive** (*bool*) –

cmd_name = setup

files

SUB_COMMAND_CLASSES

property config

List of all the commands that will be executed when running mloq setup.

Return type omegaconf.DictConfig

property sub_commands

List of all the commands that will be executed when running mloq setup.

Return type List[*mloq.command.Command*]

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

parse_config()

Update the configuration DictConfig with the Command parameters.

Return type omegaconf.DictConfig

run_side_effects()

Apply additional configuration methods.

Return type None

record_files()

Register the files that will be generated by mloq.

Return type None

17.1.1.1.2 Package Contents

Classes

CiCMD	Implement the functionality of the ci Command.
DockerCMD	Implement the functionality of the docker Command.
DocsCMD	Implement the functionality of the docs Command.
GlobalsCMD	Implement the functionality of the globals Command.
LicenseCMD	Implement the functionality of the license Command.
LintCMD	Implement the functionality of the lint Command.
PackageCMD	Implement the functionality of the package Command.
ProjectCMD	Implement the functionality of the project Command.
RequirementsCMD	Implement the functionality of the requirements Command.
SetupCMD	Implement the functionality of the setup Command.

class mloq.commands.**CiCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the ci Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name :str = ci

ubuntu_version

disable

docker

project_name

default_branch

docker_org

bot_name

bot_email

ci_python_version

python_versions

ci_extra

vendor

open_source

author

owner

email

project_url

files

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**DockerCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the docker Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = docker

files

disable

cuda

cuda_image_type

cuda_version

ubuntu_version

project_name

docker_org

python_version

base_image

test

lint

jupyter

jupyter_password

requirements

extra

makefile

static require_cuda_from_requirements(*project_config=None*)

Return True if any of the project dependencies require CUDA.

Parameters **project_config** (*Optional[omegaconf.DictConfig]*) –

Return type bool

requires_cuda()

Return True if the Docker container requires CUDA.

Return type bool

get_base_image()

Return the name of the base image for the project Docker container.

parse_config()

Update the configuration dictionary from the data entered by the user.

Return type omegaconf.DictConfig

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**DocsCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the docs Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = docs

disable

project_name

description

author

copyright_year

copyright_holder

deploy_docs

default_branch

project_url

files

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**GlobalsCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the globals Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = globals

project_name

description

author

owner

email

open_source

project_url

default_branch

license

use_poetry

main_python_version

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

parse_config()

Generate the configuration of the project via a configuration file.

Return type omegaconf.DictConfig

class mloq.commands.**LicenseCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the license Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = license

files

LICENSES

disable

license

copyright_year

copyright_holder

project_name

project_url

email

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**LintCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the lint Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = lint

files

disable

black

isort

linters

docstring_checks

pyproject_extra

project_name

makefile

poetry_requirements

ignore_files

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**PackageCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the package Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = package

files

LICENSE_CLASSIFIERS

disable

pyproject_extra

project_name

license

license_classifier

description

default_branch

project_url

owner

author

email

main_python_version

python_versions

use_poetry

parse_config()

Update the configuration DictConfig with the Command parameters.

Return type omegaconf.DictConfig

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**ProjectCMD**(*record*, *interactive=False*, ***kwargs*)

Bases: *mloq.command.Command*

Implement the functionality of the project Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

cmd_name = project

files

disable

project_name

owner

description

project_url

license

tests

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**RequirementsCMD**(*record*, *interactive=False*)

Bases: *mloq.command.Command*

Implement the functionality of the requirements Command.

Parameters

- **record** (mloq.record.CMDRecord) –
- **interactive** (*bool*) –

cmd_name = requirements

files

disable

requirements

REQUIREMENTS_ALIASES

__del__()

Remove the temporary directory when the instance is deleted.

Return type None

classmethod get_aliased_requirements_file(*option*)

Get requirement file from aliased name.

Parameters **option** (*str*) –

Return type *mloq.files.File*

classmethod read_requirements_file(*option*)

Return the content of the target requirements file form an aliased name.

Parameters **option** (*str*) –

Return type str

classmethod compose_requirements(*options*)

Return the content requirements.txt file with pinned dependencies.

The returned string contains the combined dependencies for the different options sorted alphabetically.

Parameters **options** (*Iterable[str]*) – Iterable containing the aliased names of the target dependencies for the project.

Returns str containing the pinned versions of all the selected requirements.

Return type str

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

static requirements_is_empty(*options*)

Return True if no requirements are specified for the project.

Parameters **options** (*Union[List[str], str]*) –

Return type bool

record_files()

Register the files that will be generated by mloq.

Return type None

class mloq.commands.**SetupCMD**(*record*, *interactive=False*)

Bases: *mloq.command.CommandMixin*

Implement the functionality of the setup Command.

Parameters

- **record** (mloq.record.CMDRecord) –
- **interactive** (*bool*) –

cmd_name = setup

files

SUB_COMMAND_CLASSES

property config

List of all the commands that will be executed when running mloq setup.

Return type omegaconf.DictConfig

property sub_commands

List of all the commands that will be executed when running mloq setup.

Return type List[*mloq.command.Command*]

property directories

Tuple containing Paths objects representing the directories created by the command.

Return type Tuple[pathlib.Path]

interactive_config()

Generate the configuration of the project interactively.

Return type omegaconf.DictConfig

parse_config()

Update the configuration DictConfig with the Command parameters.

Return type omegaconf.DictConfig

run_side_effects()

Apply additional configuration methods.

Return type None

record_files()

Register the files that will be generated by mloq.

Return type None

17.1.1.2 mloq.config

17.1.1.2.1 Submodules

mloq.config.configuration

This module defines the Configurable class and associated logic.

The Configurable class extends the param.Parameterizable class to keep track of the class parameters using an omegaconf.DictConfig.

Module Contents

Classes

Dataclass	Type hinting to defined a dataclass as a typing Protocol.
DictConfig	param.Parameter that defines a DictConfig object.
OmegaConfInterface	Common functionality to work with configurations.
BaseConfig	Manages getters and setters to access the target's configuration values.
Config	Config handles the *.conf* attribute of a Configurable class.
Configurable	A Configurable class is an extension of param.Parameterized that allows to handle parameters with missing values and omegaconf interpolation strings.

Functions

param_to_dataclass_dict(obj)	Create a dictionary that can be used to initialize a dataclass containing the parameters of the target param.Parameterized class.
param_to_dataclass(obj)	Create a dataclass equivalent to the target param.Parameterized target.
param_to_omegaconf(obj)	Transform a param.Parameterized class into an OmegaConf structured configuration.
is_interpolation(s)	Return True if the provided string is an OmegaConf interpolation string.
to_param_type(obj, config, key)	Transform the provided attribute of the target param.Parameterized object into the appropriate type so it can be stored in a configuration file.
to_config(config, **kwargs)	Transform the provided object into an omegaconf.DictConfig.
resolve_as_dict(obj, config, **kwargs)	Transform the provided object into a dictionary resolving all its interpolations.
safe_select(cfg, key[, default])	Access safely the target value of the provided cfg DictConfig.
as_resolved_dict(cfg)	Return a dictionary containing the resolved values for the provided DictConfig.

Attributes

DClassField
ConfigValue
ConfigurationDict
PythonType
ParamType
DataClassDict
PARAM_TO_TYPE
CONF_ATTRS

mloq.config.configuration.**DClassField**

class mloq.config.configuration.**Dataclass**

Bases: typing_extensions.Protocol

Type hinting to defined a dataclass as a typing Protocol.

__dataclass_fields__ :Dict

mloq.config.configuration.**ConfigValue**

mloq.config.configuration.**ConfigurationDict**

mloq.config.configuration.**PythonType**

mloq.config.configuration.**ParamType**

mloq.config.configuration.**DataClassDict**

class mloq.config.configuration.**DictConfig**(*default=None*, *doc=None*, *instantiate=True*, *per_instance=True*, ***kwargs*)

Bases: mloq.config.param_patch.param.ClassSelector

param.Parameter that defines a DictConfig object.

Parameters

- **default** (*Optional[omegaconf.DictConfig]*) –
- **doc** (*Optional[str]*) –
- **instantiate** (*bool*) –
- **per_instance** (*bool*) –

mloq.config.configuration.**PARAM_TO_TYPE**

mloq.config.configuration.**param_to_dataclass_dict**(*obj*)

Create a dictionary that can be used to initialize a dataclass containing the parameters of the target param.Parameterized class.

Parameters **obj** (*Union[mloq.config.param_patch.param.Parameterized, Any]*) – Class or instance of a param.Parameterized class.

Returns dict containing the fields required to define a dataclass with the obj parameters.

Return type Dict[str, Tuple[type, DClassField]]

mloq.config.configuration.**param_to_dataclass**(*obj*)

Create a dataclass equivalent to the target param.Parameterized target.

Parameters **obj** (*Union[mloq.config.param_patch.param.Parameterized, Any]*) –

Return type type

mloq.config.configuration.**param_to_omegaconf**(*obj*)

Transform a param.Parameterized class into an OmegaConf structured configuration.

Parameters **obj** (*Union[mloq.config.param_patch.param.Parameterized, Any]*) –

Return type omegaconf.DictConfig

mloq.config.configuration.**is_interpolation**(*s*)

Return True if the provided string is an OmegaConf interpolation string.

Parameters **s** (*str*) –

Return type bool

mloq.config.configuration.**to_param_type**(*obj*, *config*, *key*)

Transform the provided attribute of the target param.Parameterized object into the appropriate type so it can be stored in a configuration file.

Parameters

- **obj** (*mloq.config.param_patch.param.Parameterized*) –
- **config** (DictConfig) –
- **key** (*str*) –

Return type Any

mloq.config.configuration.**to_config**(*config*, ***kwargs*)

Transform the provided object into an omegaconf.DictConfig.

Parameters **config** (*Union[omegaconf.DictConfig, ConfigurationDict, Dataclass, mloq.config.param_patch.param.Parameterized, None]*) –

Return type omegaconf.DictConfig

mloq.config.configuration.**resolve_as_dict**(*obj*, *config*, ***kwargs*)

Transform the provided object into a dictionary resolving all its interpolations.

Parameters **config** (*Union[omegaconf.DictConfig, ConfigurationDict, Dataclass, mloq.config.param_patch.param.Parameterized]*) –

Return type ConfigurationDict

mloq.config.configuration.**safe_select**(*cfg*, *key*, *default=None*)

Access safely the target value of the provided cfg DictConfig.

Return MISSING if the value cannot be resolved or it's missing.

Parameters

- **cfg** (DictConfig) –
- **key** (*str*) –
- **default** (*Any*) –

Return type Any

mloq.config.configuration.**as_resolved_dict**(*cfg*)

Return a dictionary containing the resolved values for the provided DictConfig.

Parameters **cfg** (DictConfig) –

Return type ConfigurationDict

class mloq.config.configuration.**OmegaConfInterface**(*target*, *allow_missing=False*)

Common functionality to work with configurations.

Parameters

- **target** (Configurable) –
- **allow_missing** (*bool*) –

property config

Return a DictConfig containing the target configuration.

Return type omegaconf.DictConfig

property interpolations

Return a dictionary containing the interpolations of the target configuration.

Return type ConfigurationDict

property missing

Return a list containing the names of the configuration that are MISSING.

Return type List[Union[str, int, enum.Enum, float, bool]]

_resolve_inplace(*key=None*)

Resolve and update the target attribute if it's an interpolation string.

Parameters **key** (*Optional[str]*) –

Return type None

resolve(*key=None*, *inplace=False*)

Resolve the target attribute if it is an interpolation string.

Parameters

- **key** (*Optional[str]*) – Name of the target's attribute to be resolved.
- **inplace** (*bool*) – If True, update the configuration value replacing the interpolation string with the resolved value.

Returns Resolved value of the target attribute.

Return type Union[omegaconf.Container, ConfigValue, None]

is_missing(*key*)

Return True if the key target's attribute is Missing, otherwise return False.

Parameters **key** (*str*) –

Return type bool

is_interpolation(*key*)

Check if the key target's attribute is an interpolation string.

Parameters **key** (*str*) –

Return type bool

select(*key*, *default=None*)

Select the key target's attribute.

Return MISSING if key corresponds to a missing value, or an interpolation that resolves to a missing value.

Return type Any

class mloq.config.configuration.**BaseConfig**(*target*, *config=None*, *cfg_node=None*, *allow_missing=False*, ***kwargs*)

Bases: *OmegaConfInterface*

Manages getters and setters to access the target's configuration values.

Parameters

- **target** (Configurable) –
- **config** (*Optional[Union[ConfigurationDict, omegaconf.DictConfig]]*) –
- **cfg_node** (*Optional[str]*) –
- **allow_missing** (*bool*) –

__getitem__(*item*)

Access the target config value.

Parameters **item** (*str*) –

Return type Any

__setitem__(*key*, *value*)

Set the target config value.

Parameters **key** (*str*) –

Return type Any

to_container(*resolve=False*, ***kwargs*)

Return a container containing the target's configuration.

Parameters **resolve** (*bool*) –

Return type omegaconf.Container

static _resolve_node(*kwargs*, *config=None*, *cfg_node=None*)

Return a DictConfig containing the resolved configuration values defined in kwargs.

Parameters

- **kwargs** (*dict*) –
- **config** (*Optional[omegaconf.DictConfig]*) –
- **cfg_node** (*Optional[str]*) –

Return type omegaconf.DictConfig

_setup_config(*config=None*, *cfg_node=None*, ***kwargs*)

Initialize and validate the structured config of target.

Parameters

- **config** (*Optional[Union[ConfigurationDict, omegaconf.DictConfig]]*) –
- **cfg_node** (*Optional[str]*) –

class mloq.config.configuration.**Config**(*target*, *config=None*, *cfg_node=None*, *allow_missing=False*, ***kwargs*)

Bases: *BaseConfig*

Config handles the *.conf* attribute of a Configurable class.

It is analogous to *.param* for param.Parameterized classes. This class implements all the logic to access and update the config attribute of a Configurable class, which returns a DictConfig instance that is automatically update when the parameters of the class change.

Parameters

- **target** (*Configurable*) –
- **config** (*Optional[Union[ConfigurationDict, omegaconf.DictConfig]]*) –
- **cfg_node** (*Optional[str]*) –
- **allow_missing** (*bool*) –

property params

Return the param.Parameter dictionary of the target configurable.

Return type Dict[str, mloq.config.param_patch.param.Parameter]

resolve(*key=None*, *inplace=False*)

Resolve the key attribute of the target Configurable.

Parameters

- **key** (*Optional[str]*) –
- **inplace** (*bool*) –

Return type Union[omegaconf.Container, ConfigValue, None]

to_param_type(*key*)

Transform the value of the key target's parameter to a DictConfig compatible type.

Return type Any

dataclass_dict(*ignore=None*)

Return a dictionary to create a dataclass with the target's parameters.

Parameters **ignore** (*Optional[Union[list, set, tuple, str]]*) –

Return type DataClassDict

to_dataclass()

Return a dataclass describing the parameter values of the target Configurable.

Return type type

to_dictconfig()

Return a structured DictConfig containing the parameters of the target Configurable.

Return type *DictConfig*

sync()

Ensure the parameter values of the target class have the right type.

_setup_config(*config=None*, *cfg_node=None*, ***kwargs*)

Initialize and validate the structured config of target.

Parameters

- **config** (*Optional[Union[ConfigurationDict, DictConfig]]*) –
- **cfg_node** (*Optional[str]*) –

mloq.config.configuration.**CONF_ATTRS**

class mloq.config.configuration.**Configurable**(*config=None*, *throw_on_missing=True*, *cfg_node=None*, ***kwargs*)

Bases: mloq.config.param_patch.param.Parameterized

A Configurable class is an extension of param.Parameterized that allows to handle parameters with missing values and omegaconf interpolation strings.

It add a config attribute containing an omegaconf.DictConfig that contains the values of the class param.Parameters.

It also provides a *conf* attribute that allows to access omegaconf functionality for managing configurations in a similar fashion as the *param* attribute allows to access param.Parameter functionality.

Parameters

- **config** (*Optional[Union[ConfigurationDict, DictConfig]]*) –
- **throw_on_missing** (*bool*) –
- **cfg_node** (*Optional[str]*) –

config

property conf

Access the Config instance that tracks and manages the values in the class config.

Return type *Config*

__setattr__(*key*, *value*)

Update the config values when setting a parameter.

__getattr__(*item*)

Add support for MISSING values when accessing the parameter values.

mloq.config.custom_click

This is mostly a copy paste from https://github.com/pallets/click/blob/2fc486c880eda9fdb746ed8baa49416acab9ea6d/src/click/termui.py

Modified to allow prompt input that has a different color than the prompt text, while keeping the color of the default prompt values the same as the prompt text color.

Module Contents

Functions

hidden_prompt_func(prompt)	Input hidden text from the user.
_build_prompt(text, suffix[, show_default, default, ...])	
_format_default(default)	
prompt(text[, default, hide_input, ...])	Prompts a user for input. This is a convenience function that can be used to prompt a user for input later.
confirm(text[, default, abort, prompt_suffix, ...])	Prompts for confirmation (yes/no question).

Attributes

visible_prompt_func
_ansi_reset_all

mloq.config.custom_click.**visible_prompt_func**

mloq.config.custom_click._ansi_reset_all = [0m

mloq.config.custom_click.**hidden_prompt_func**(*prompt*)

Input hidden text from the user.

mloq.config.custom_click.**_build_prompt**(*text*, *suffix*, *show_default=False*, *default=None*, *show_choices=True*, *type=None*)

mloq.config.custom_click.**_format_default**(*default*)

mloq.config.custom_click.**prompt**(*text*, *default=None*, *hide_input=False*, *confirmation_prompt=False*, *type=None*, *value_proc=None*, *prompt_suffix=': '*, *show_default=True*, *err=False*, *show_choices=True*)

Prompts a user for input. This is a convenience function that can be used to prompt a user for input later.

If the user aborts the input by sending a interrupt signal, this function will catch it and raise a `Abort` exception.

New in version 7.0: Added the show_choices parameter.

New in version 6.0: Added unicode support for cmd.exe on Windows.

New in version 4.0: Added the *err* parameter.

Parameters

- **text** – the text to show for the prompt.
- **default** – the default value to use if no input happens. If this is not given it will prompt until it's aborted.
- **hide_input** – if this is set to true then the input value will be hidden.
- **confirmation_prompt** – asks for confirmation for the value.
- **type** – the type to use to check the value against.
- **value_proc** – if this parameter is provided it's a function that is invoked instead of the type conversion to convert a value.
- **prompt_suffix** – a suffix that should be added to the prompt.
- **show_default** – shows or hides the default value in the prompt.
- **err** – if set to true the file defaults to `stderr` instead of `stdout`, the same as with echo.
- **show_choices** – Show or hide choices if the passed type is a Choice. For example if type is a Choice of either day or week, show_choices is true and text is "Group by" then the prompt will be "Group by (day, week): ".

Returns None

Return type None

mloq.config.custom_click.**confirm**(*text*, *default=False*, *abort=False*, *prompt_suffix=': '*, *show_default=True*, *err=False*)

Prompts for confirmation (yes/no question).

If the user aborts the input by sending a interrupt signal this function will catch it and raise a `Abort` exception.

New in version 4.0: Added the *err* parameter.

Parameters

- **text** – the question to ask.
- **default** – the default for the prompt.

- **abort** – if this is set to *True* a negative answer aborts the exception by raising `Abort`.
- **prompt_suffix** – a suffix that should be added to the prompt.
- **show_default** – shows or hides the default value in the prompt.
- **err** – if set to true the file defaults to `stderr` instead of `stdout`, the same as with echo.

Returns User's decision.

Return type bool

mloq.config.param_patch

Patch param to allow omegaconf Missing values and interpolation strings.

Module Contents

Classes

__ParamPatcher	Patch the param package to handle missing values and interpolation strings.

Functions

__init__patched(self[, default])	Handle missing values and interpolations trings when initializing a param.Parameter.
_create_param__(item)	Patch the target param.Parameter to support missing values and interpolations strings.

Attributes

__DEFAULT_MARK
PATCHED_PARAMETERS
param

mloq.config.param_patch.**__DEFAULT_MARK = __DEFAULT_MARK__**

mloq.config.param_patch.**__init__patched**(*self*, *default=__DEFAULT_MARK*, ***kwargs*)

Handle missing values and interpolations trings when initializing a param.Parameter.

mloq.config.param_patch.**_create_param__**(*item*)

Patch the target param.Parameter to support missing values and interpolations strings.

mloq.config.param_patch.**PATCHED_PARAMETERS**

class mloq.config.param_patch.**__ParamPatcher**

Patch the param package to handle missing values and interpolation strings.

PATCHED_PARAMETERS

__getattr__(*item*)

Patch the parameters included in PATCHED_PARAMETERS.

abstract __setattr__(*key*, *value*)

Read only monkeypatching.

mloq.config.param_patch.**param**

mloq.config.prompt

This file contains the logic defining all the parameters needed to set up a project with mloq.

Module Contents

Classes

PromptParam	Defines a configuration parameter.
MultiChoicePrompt	Define a configuration parameter that can take multiple values from a pre-defined set of values.
StringPrompt	Define a configuration parameter that can take a string value.
IntPrompt	Define a configuration parameter that can take an integer value.
FloatPrompt	Define a configuration parameter that can take a floating point value.
BooleanPrompt	Defines a boolean configuration parameter.
Prompt	Manage all the functionality needed to display a cli prompt.
Promptable	Configurable class that allows to define the parameter values interactively using CLI prompts.

Attributes

Choices
PARAM_TO_PROMPT

mloq.config.prompt.**Choices**

class mloq.config.prompt.**PromptParam**(*name*, *target*, ***kwargs*)

Defines a configuration parameter.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –

property **param**

Get the param.Parameter object corresponding to the current configuration parameter.

Return type param.Parameter

property **value**

Return the value of the configuration parameter.

Return type Any

property **config**

Return the value of the parameter as defined in its config DictConfig.

Return type Any

__call__(*interactive=False*, *default=None*, ***kwargs*)

Return the value of the parameter parsing it from the different input sources available.

Parameters

- **interactive** (*bool*) – Prompt the user to input the value from CLI if it's not defined in config or as en environment variable.
- **default** (*Optional[Any]*) – Default value displayed in the interactive mode.
- ****kwargs** – Passed to click.prompt in interactive mode. Overrides the values defined in __init__

Returns Value of the parameter.

_prompt(*value*, ***kwargs*)

Prompt user for value.

class mloq.config.prompt.**MultiChoicePrompt**(*name*, *target*, *choices=None*, ***kwargs*)

Bases: *PromptParam*

Define a configuration parameter that can take multiple values from a pre-defined set of values.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –
- **choices** (*Optional[Choices]*) –

_prompt(*value*, ***kwargs*)

Transform the parsed string from the CLI into a list of selected values.

Return type List[str]

static _parse_string(*value*)

Return type List[str]

class mloq.config.prompt.**StringPrompt**(*name*, *target*, ***kwargs*)

Bases: *PromptParam*

Define a configuration parameter that can take a string value.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –

class mloq.config.prompt.**IntPrompt**(*name*, *target*, ***kwargs*)

Bases: *PromptParam*

Define a configuration parameter that can take an integer value.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –

class mloq.config.prompt.**FloatPrompt**(*name*, *target*, ***kwargs*)

Bases: *PromptParam*

Define a configuration parameter that can take a floating point value.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –

class mloq.config.prompt.**BooleanPrompt**(*name*, *target*, ***kwargs*)

Bases: *PromptParam*

Defines a boolean configuration parameter.

It allows to parse a configuration value from different sources in the following order:

1. Environment variable named as MLOQ_PARAM_NAME
2. Values defined in mloq.yaml
3. Interactive promp from CLI (Optional)

Parameters

- **name** (*str*) –
- **target** (mloq.config.configuration.Configurable) –

_prompt(*value*, ***kwargs*)

Prompt user for value.

mloq.config.prompt.**PARAM_TO_PROMPT**

class mloq.config.prompt.**Prompt**(*target*)

Manage all the functionality needed to display a cli prompt.

It allows to interactively define the values of the different parameters of a class.

Parameters **target** (Promptable) –

__call__(*key*, *inplace=False*, ***kwargs*)

Display the a prompt to interactively define the parameter values of target.

Parameters

- **key** (*str*) –
- **inplace** (*bool*) –

Return type Any

_init_prompts()

Initialize the prompts corresponding to the target Promptable parameters.

Return type None

prompt(*key*, *inplace=False*, ***kwargs*)

Display the a prompt to interactively define the parameter values of target.

Parameters

- **key** (*str*) –
- **inplace** (*bool*) –

Return type Any

prompt_all(*inplace=False*, ***kwargs*)

Prompt all the target's parameters.

Return a dictionary containing the provided values.

Parameters **inplace** (*bool*) –

Return type Dict[str, Any]

class mloq.config.prompt.**Promptable**(***kwargs*)

Bases: *mloq.config.configuration.Configurable*

Configurable class that allows to define the parameter values interactively using CLI prompts.

It contains a prompt attribute in charge of managing the prompting functionality for the param.Parameters defined.

17.1.2 Submodules

17.1.2.1 mloq.__main__

Command line interface for mloq.

17.1.2.2 mloq._utils

This module contains some utilities that are not currently used.

17.1.2.2.1 Module Contents

Functions

dir_trees_are_equal(dir1, dir2)	Compare two directories recursively. Files in each directory are assumed to be equal if their names and contents are equal.
files_are_equal(path1, path2)	Compare the content of two files.
get_generated_files(path)	List all the files generated in the last mloq run.
get_generated_directories(path)	List all the directories generated in the last mloq run.
check_directories_exist(paths)	Check if the provided paths exist.
get_docker_python_version(template)	Return the highest python version defined for the project.
write_config_setup(config, path[, safe])	Write setup config in a yaml file.
load_empty_config_setup()	Return a dictionary containing all the MLOQ setup config values set to None.

mloq._utils.**dir_trees_are_equal**(*dir1*, *dir2*)

Compare two directories recursively. Files in each directory are assumed to be equal if their names and contents are equal.

@param dir1: First directory path @param dir2: Second directory path

@**return: True if the directory trees are the same and** there were no errors while accessing the directories or files, False otherwise.

Parameters

- **dir1** (*Union[str, pathlib.Path]*) –
- **dir2** (*Union[str, pathlib.Path]*) –

Return type bool

mloq._utils.**files_are_equal**(*path1*, *path2*)

Compare the content of two files.

Compare two incoming files. They are assumed equal if their contents are the same.

Parameters

- **path1** (*Union[str, pathlib.Path]*) – Path containing the first file to be compared.
- **path2** (*Union[str, pathlib.Path]*) – Path containing the second file to be compared.

Returns

It returns True if the two given files are equal and no errors have arisen during the process. False otherwise.

Return type bool

mloq._utils.**get_generated_files**(*path*)

List all the files generated in the last mloq run.

Parameters **path** (*Union[str, pathlib.Path]*) – path to WHAT_MLOQ_GENERATED.md file.

Returns List of Path containing the names of the files generated by mloq.

Return type List[pathlib.Path]

mloq._utils.**get_generated_directories**(*path*)

List all the directories generated in the last mloq run.

Parameters **path** (*Union[str, pathlib.Path]*) – path to WHAT_MLOQ_GENERATED.md file.

Returns List of Path containing the names of the directories generated by mloq.

Return type List[pathlib.Path]

mloq._utils.**check_directories_exist**(*paths*)

Check if the provided paths exist.

Parameters **paths** (*List[Union[str, pathlib.Path]]*) – List of paths that will be checked

Returns True if all the provided paths exist. False otherwise.

Return type bool

mloq._utils.**get_docker_python_version**(*template*)

Return the highest python version defined for the project.

Parameters **template** (*omegaconf.DictConfig*) –

Return type str

mloq._utils.**write_config_setup**(*config*, *path*, *safe=False*)

Write setup config in a yaml file.

Parameters

- **config** (*omegaconf.DictConfig*) –
- **path** (*Union[pathlib.Path, str]*) –
- **safe** (*bool*) –

mloq._utils.**load_empty_config_setup**()

Return a dictionary containing all the MLOQ setup config values set to None.

Return type omegaconf.DictConfig

17.1.2.3 mloq.cli

Command line interface for mloq.

17.1.2.3.1 Module Contents

Classes

MloqCLI	Load the commands available at runtime from the files present in the command module.

Functions

mloq_click_command(func)	Wrap a command function to interface with click.
cli()	
welcome_message([extra, string])	Welcome message to be displayed during interactive setup.

Attributes

overwrite_opt
config_file_opt
only_config_opt
output_directory_arg
interactive_opt
hydra_args

mloq.cli.**overwrite_opt**

mloq.cli.**config_file_opt**

mloq.cli.**only_config_opt**

mloq.cli.**output_directory_arg**

mloq.cli.**interactive_opt**

mloq.cli.**hydra_args**

mloq.cli.**mloq_click_command**(*func*)

Wrap a command function to interface with click.

class mloq.cli.**MloqCLI**(*name=None*, *invoke_without_command=False*, *no_args_is_help=None*, *subcommand_metavar=None*, *chain=False*, *result_callback=None*, ***attrs*)

Bases: `click.MultiCommand`

Load the commands available at runtime from the files present in the command module.

Parameters

- **name** (*Optional[str]*) –
- **invoke_without_command** (*bool*) –
- **no_args_is_help** (*Optional[bool]*) –
- **subcommand_metavar** (*Optional[str]*) –
- **chain** (*bool*) –
- **result_callback** (*Optional[Callable[Ellipsis, Any]]*) –
- **attrs** (*Any*) –

command_folder

list_commands(*ctx*)

List the names of the mloq commands available.

get_command(*ctx*, *name*)

Create the command callable corresponding to the provided command name.

Return type Callable

mloq.cli.**cli**()

mloq.cli.**welcome_message**(*extra=False*, *string=None*)

Welcome message to be displayed during interactive setup.

Parameters

- **extra** (*bool*) –
- **string** (*Optional[str]*) –

17.1.2.4 mloq.command

This module defines the base Command class used for defining mloq commands.

17.1.2.4.1 Module Contents

Classes

CommandMixin	Class containing the interface for defining an MLOQ Command.
Command	Define blueprints for generating custom mloq commands.

class mloq.command.**CommandMixin**(*record*, *interactive=False*, **args*, ***kwargs*)

Class containing the interface for defining an MLOQ Command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

files :tuple

cmd_name = command

property record

Return a CMDRecord that keeps track of the files and directories the command creates.

Return type mloq.writer.CMDRecord

property directories

Tuple containing Paths objects representing the directories the Command creates.

Override this property if your command creates any directories.

Returns Tuple of `Path` objects representing the path to the directories that the *Command* will create.

Return type Tuple[pathlib.Path]

parse_config()

Update the configuration dictionary from the data entered by the user.

Given the basic configuration skeleton (contained in mloq.yaml), this method updates the values of those parameters (included in CONFIG object) that are related to the selected command. Incoming values are introduced either interactively or via a custom user's mloq.yaml file.

Returns It returns an updated version of the 'config' attribute of the 'record' instance.

Return type omegaconf.DictConfig

abstract interactive_config()

Pass user's configuration interactively.

Return type omegaconf.DictConfig

record_files()

Register the files that will be generated by mloq.

Return type None

`record_directories()`

Register the directories that will be generated by mloq.

Return type None

`configure()`

Save the updated version of the 'config' attribute.

After parsing the new configuration values introduced by the user, this method registers and saves this updated configuration within the '_config' attribute of the 'record' instance.

Return type None

`run_side_effects()`

Apply additional configuration methods.

Return type None

`run()`

Record the files and directories generated by mloq according to the user's configuration.

This method updates the configuration dictionary with the values introduced by the user. Once the parameters have been revised, the files and directories that will be generated by mloq are registered within the 'record' instance.

Returns

It returns an updated version of the CMDRecord instance, where the files and directories that will be generated by mloq are recorded within the 'record' instance.

Return type mloq.writer.CMDRecord

`class mloq.command.Command(record, interactive=False, **kwargs)`

Bases: *CommandMixin*, *mloq.config.prompt.Promptable*

Define blueprints for generating custom mloq commands.

Base class used for defining new mloq commands. It establishes the fundamental methods for defining and updating the configuration values used to create the necessary files for the user's project, while registering the latter for later use.

This class is initialized from a CMDRecord instance, object where the user's configuration, as well as the files and directories that will be generated, are stored.

class Attributes: name: Name of the command. files: Tuple containing the templates used for creating the necessary files of your project. CONFIG: NamedTuple containing the keys and values of your configuration RELATIVE to the command.

Parameters

- **record** (*mloq.writer.CMDRecord*) –
- **interactive** (*bool*) –

`interactive_config()`

Pass user's configuration interactively.

Return type omegaconf.DictConfig

`_config_from_record()`

Return type omegaconf.DictConfig

17.1.2.5 mloq.custom_click

This is mostly a copy paste from https://github.com/pallets/click/blob/2fc486c880eda9fdb746ed8baa49416acab9ea6d/src/click/termui.py

Modified to allow prompt input that has a different color than the prompt text, while keeping the color of the default prompt values the same as the prompt text color.

17.1.2.5.1 Module Contents

Functions

hidden_prompt_func(prompt)	Input hidden text from the user.
_build_prompt(text, suffix[, show_default, default, ...])	
_format_default(default)	
prompt(text[, default, hide_input, ...])	Prompts a user for input. This is a convenience function that can be used to prompt a user for input later.
confirm(text[, default, abort, prompt_suffix, ...])	Prompts for confirmation (yes/no question).

Attributes

visible_prompt_func	
_ansi_reset_all	

mloq.custom_click.**visible_prompt_func**

mloq.custom_click.**_ansi_reset_all = [0m**

mloq.custom_click.**hidden_prompt_func**(*prompt*)

Input hidden text from the user.

mloq.custom_click.**_build_prompt**(*text*, *suffix*, *show_default=False*, *default=None*, *show_choices=True*, *type=None*)

mloq.custom_click.**_format_default**(*default*)

mloq.custom_click.**prompt**(*text*, *default=None*, *hide_input=False*, *confirmation_prompt=False*, *type=None*, *value_proc=None*, *prompt_suffix=': '*, *show_default=True*, *err=False*, *show_choices=True*)

Prompts a user for input. This is a convenience function that can be used to prompt a user for input later.

If the user aborts the input by sending a interrupt signal, this function will catch it and raise a `Abort` exception.

New in version 7.0: Added the show_choices parameter.

New in version 6.0: Added unicode support for cmd.exe on Windows.

New in version 4.0: Added the *err* parameter.

Parameters

- **text** – the text to show for the prompt.
- **default** – the default value to use if no input happens. If this is not given it will prompt until it's aborted.
- **hide_input** – if this is set to true then the input value will be hidden.
- **confirmation_prompt** – asks for confirmation for the value.
- **type** – the type to use to check the value against.
- **value_proc** – if this parameter is provided it's a function that is invoked instead of the type conversion to convert a value.
- **prompt_suffix** – a suffix that should be added to the prompt.
- **show_default** – shows or hides the default value in the prompt.
- **err** – if set to true the file defaults to `stderr` instead of `stdout`, the same as with echo.
- **show_choices** – Show or hide choices if the passed type is a Choice. For example if type is a Choice of either day or week, show_choices is true and text is "Group by" then the prompt will be "Group by (day, week): ".

Returns None

Return type None

`mloq.custom_click.`**confirm**(*text*, *default=False*, *abort=False*, *prompt_suffix=': '*, *show_default=True*, *err=False*)

Prompts for confirmation (yes/no question).

If the user aborts the input by sending a interrupt signal this function will catch it and raise a `Abort` exception.

New in version 4.0: Added the *err* parameter.

Parameters

- **text** – the question to ask.
- **default** – the default for the prompt.
- **abort** – if this is set to *True* a negative answer aborts the exception by raising `Abort`.
- **prompt_suffix** – a suffix that should be added to the prompt.
- **show_default** – shows or hides the default value in the prompt.
- **err** – if set to true the file defaults to `stderr` instead of `stdout`, the same as with echo.

Returns User's decision.

Return type bool

17.1.2.6 mloq.failure

Define Failure - the exception to raise when we break.

17.1.2.6.1 Module Contents

exception mloq.failure.**Failure**

Bases: Exception

Raised when a project setup critical error happens.

__str__()

Delegate __str__ to the underlying cause if it exists.

Return type str

exception mloq.failure.**MissingConfigValue**

Bases: *Failure*

Raised when a parameter is not defined in the mloq DictConfig.

17.1.2.7 mloq.files

This module defines all the different assets accessible from mloq.

17.1.2.7.1 Module Contents

Classes

File	Generates project files.

Functions

file(name, path[, description, dst, is_static])	Define a new asset as a File namedtuple.
read_file(file)	Return and string with the content of the provided file.

Attributes

ASSETS_PATH
SHARED_ASSETS_PATH
MLOQ_ASSETS_PATH
REQUIREMENTS_PATH
mloq_yml
what_mloq_generated
requirements
dogfood_req
makefile
pyproject_toml

class mloq.files.**File**

Bases: NamedTuple

Generates project files.

This class defines templating files, which will be rendered according to the user's configuration. Besides, File instances have additional attributes used for specifying the destination of the generated file.

Attributes of this class: name: Name of the templating file. src: Location of the templating file. dst: Name of the file generated from the templating file. description: Short description of the current file. is_static: Boolean value. If True, the templating file does not

admit render parameters.

name :str

src :pathlib.Path

dst :pathlib.Path

description :str

is_static :bool

mloq.files.**file**(*name*, *path*, *description=None*, *dst=None*, *is_static=False*)

Define a new asset as a File namedtuple.

Parameters

- **name** (*str*) –
- **path** (*Union[pathlib.Path, str]*) –
- **description** (*Optional[str]*) –

- **dst** (*Optional[Union[pathlib.Path, str]]*) –
- **is_static** (*bool*) –

Return type *File*

mloq.files.**ASSETS_PATH**

mloq.files.**SHARED_ASSETS_PATH**

mloq.files.**MLOQ_ASSETS_PATH**

mloq.files.**REQUIREMENTS_PATH**

mloq.files.**mloq_yml**

mloq.files.**what_mloq_generated**

mloq.files.**requirements**

mloq.files.**dogfood_req**

mloq.files.**makefile**

mloq.files.**pyproject_toml**

mloq.files.**read_file**(*file*)

Return and string with the content of the provided file.

Parameters **file** (File) –

Return type str

17.1.2.8 mloq.git

Setup Git repository for the project.

17.1.2.8.1 Module Contents

Functions

setup_git(path, config)	Initialize a Git repository over the generated files.
_git_cmd(git_dir, *parts)	

mloq.git.**setup_git**(*path*, *config*)

Initialize a Git repository over the generated files.

Parameters

- **path** (*Union[pathlib.Path, str]*) –
- **config** (*omegaconf.DictConfig*) –

Return type None

mloq.git.**_git_cmd**(*git_dir*, **parts*)

Parameters

- **git_dir** (*str*) –
- **parts** (*str*) –

Return type None

17.1.2.9 mloq.record

This module contains the classes that keep track of the internal state of the application when running a Command.

17.1.2.9.1 Module Contents

Classes

Ledger	Keep track of the generated files.
CMDRecord	Keep track of files and directories that will be created by mloq.

class mloq.record.**Ledger**

Keep track of the generated files.

property files

Return the list of generated file names.

Return type List[Tuple[str, str]]

register(*file*, *description=None*)

Append another generated file to the book.

Parameters

- **file** (*Union[mloq.files.File, str, pathlib.Path]*) –
- **description** (*Optional[str]*) –

Return type None

class mloq.record.**CMDRecord**(*config=None*, *files=None*, *directories=None*)

Keep track of files and directories that will be created by mloq.

The *CMDRecord* acts as a single source of truth for storing the files and directories generated by *mloq* as well as the necessary configuration to generate them.

This class registers three separate data sources:

- ***config***: An `omegaconf.DictConfig` that contains the configuration of all the commands executed by *mloq*.
- ***files***: A *dict* containing the content and location of the files generated by mloq. It is indexed by `Path` objects that indicate where the file will be created, and its values are instances of *mloq.files.File*.
- ***directories***: List of `Path` instances pointing to the different directories that *mloq* will create.

This class is initialized from a configuration dictionary. The dictionary can be either an `omegaconf.DictConfig` or an empty dictionary.

Parameters

- **config** (*Optional[omegaconf.DictConfig]*) –
- **files** (*Optional[Dict[pathlib.Path, mloq.files.File]]*) –
- **directories** (*Optional[List[pathlib.Path]]*) –

property config

Store the configuration parameters that govern the project's structure.

It contains one configuration entry per each Command that will be run, and each entry will only be modified by the Command it represents. Each Command instance is responsible for updating its corresponding *config* values.

Returns omegaconf.DictConfig that contains the configuration of all the commands executed by *mloq*.

Return type omegaconf.DictConfig

property files

Return the dictionary of files used by mloq to generate the project configuration.

files is a *dict* containing the content and location of the files generated by mloq. It is indexed by Path objects that indicate where the file will be created, and its values are instances of *mloq.files.File*.

Each different Command is responsible for registering the files it generates to the *files* dictionary.

Return type Dict[pathlib.Path, *mloq.files.File*]

property directories

Contain the folders that will be created by mloq for storing the project's files.

Return type List[pathlib.Path]

update_config(*config*)

Update the configuration attribute according to the values entered by the user.

Parameters **config** (*omegaconf.DictConfig*) –

Return type None

register_file(*file*, *path*, *description=None*)

Append a new file to the 'files' container.

Keys are Path strings describing the location where the file will be created. Values are File objects containing the information of the file that will be generated.

Parameters

- **file** (mloq.files.File) –
- **path** (*Union[pathlib.Path, str]*) –
- **description** (*Optional[str]*) –

Return type None

register_directory(*path*)

Append a new directory path to the 'directories' container.

Parameters **path** (*Union[pathlib.Path, str]*) –

Return type None

17.1.2.10 mloq.runner

This module defines the pipeline for running a mloq command, such as config loading, template writing and interfacing with click.

17.1.2.10.1 Module Contents

Functions

load_config(config_file, hydra_args)	Load the necessary configuration for running mloq from a mloq.yaml file.
write_record(record, path[, overwrite, only_config])	Write the contents of the provided record to the target path.
run_command(cmd_cls[, use_click])	Run the given Command class.

mloq.runner.**load_config**(*config_file*, *hydra_args*)

Load the necessary configuration for running mloq from a mloq.yaml file.

If no path to mloq.yaml is provided, it returns a template to be filled in using the interactive mode.

Parameters

- **config_file** (*Union[pathlib.Path, str]*) – Path to the target mloq.yaml file.
- **hydra_args** (*str*) – Arguments passed to hydra for composing the project configuration.

Returns DictConfig containing the project configuration.

Return type omegaconf.DictConfig

mloq.runner.**write_record**(*record*, *path*, *overwrite=False*, *only_config=False*)

Write the contents of the provided record to the target path.

The writing process is performed by :class: *Writer*, class that fills in rendered templates according to the given configuration.

Parameters

- **record** (mloq.record.CMDRecord) – CMDRecord containing all the data to be written.
- **path** (*Union[pathlib.Path, str]*) – Target directory to write the data.
- **overwrite** (*bool*) – If True overwrite existing files.
- **only_config** (*bool*) – Do not write any file except mloq.yaml

Returns None.

Return type None

mloq.runner.**run_command**(*cmd_cls*, *use_click=True*)

Run the given Command class.

Parameters

- **cmd_cls** – Command to be executed.
- **use_click** (*bool*) – If True run the function as a "cli" command.

Returns A function that will run the target class as a mloq command.

Return type Callable

17.1.2.11 mloq.templating

This module defines common functionality for rendering and writing File templates.

17.1.2.11.1 Module Contents

Functions

render_template(file, kwargs)	Render a jinja template with the provided parameter dict.
write_template(file, config, path, ledger[, overwrite])	Create new file containing the rendered template found in source_path.

Attributes

jinja_env

mloq.templating.**jinja_env**

mloq.templating.**render_template**(*file*, *kwargs*)

Render a jinja template with the provided parameter dict.

Parameters

- **file** (mloq.files.File) – File object representing the jinja template that will be rendered.
- **kwargs** (*Mapping[str, Any]*) – Dictionary containing the parameters key and corresponding values that will be used to render the template.

Returns String containing the rendered template.

Return type str

mloq.templating.**write_template**(*file*, *config*, *path*, *ledger*, *overwrite=False*)

Create new file containing the rendered template found in source_path.

Parameters

- **file** (mloq.files.File) – File object representing the jinja template that will be rendered.
- **config** (*omegaconf.DictConfig*) – OmegaConf dictionary containing the parameters key and corresponding values that will be used to render the templates.
- **path** (*Union[pathlib.Path, str]*) – Absolute path to the folder where the file will be written.
- **ledger** (mloq.record.Ledger) – Book keeper to keep track of the generated files.
- **overwrite** (*bool*) – If False, copy the file if it does not already exists in the target path. If True, overwrite the target file if it is already present.

Returns None.

17.1.2.12 mloq.version

Current version of the project. Do not modify manually.

17.1.2.12.1 Module Contents

mloq.version.**__version__ = 0.0.75**

17.1.2.13 mloq.writer

The writer module defines the Writer class, which is in charge of creating the files and directories specified in the CMDRecord.

17.1.2.13.1 Module Contents

Classes

Writer	Write all the files specified on the provided CMDRecord.

class mloq.writer.**Writer**(*path*, *record*, *overwrite=False*)

Write all the files specified on the provided CMDRecord.

This class fills in rendered templates according to the provided configuration and generates the resulting file on the specified folder.

Attributes of this class:

path: Path string describing the destination folder where the file will be created.

ledger: Instance of the Ledger class. It contains a dictionary summarizing the files that will by generated by mloq.

record: Instance of the CMDRecord class. It keeps track of all files and directories that will be created from the user's configuration.

overwrite: Boolean value. If True, existing files will be rewritten by mloq application.

Parameters

- **path** (*Union[pathlib.Path, str]*) –
- **record** (mloq.record.CMDRecord) –
- **overwrite** (*bool*) –

property **path**

Path string describing the location where the files will be generated.

Return type pathlib.Path

property **ledger**

Keep track of the generated files.

Return type *mloq.record.Ledger*

property record

Register the files and directories generated by the mloq application.

Return type *mloq.record.CMDRecord*

create_directories()

Create the folders registered inside the attribute 'record.directories'.

Return type None

write_templates()

Generate the files recorded in the attribute 'record.files' on the specified path.

Return type None

dump_ledger()

Write the summary of the generated files.

This method collects the elements stored in ledger to create a markdown document that summarizes all the files generated by the MLOQ application.

Return type None

write_template(*file*, *path*, *config*)

Create new file containing the rendered template configuration.

Parameters

- **file** (mloq.files.File) – File object representing the jinja template that will be rendered.
- **path** (*pathlib.Path*) – Target folder where the generated files will be written.
- **config** (*omegaconf.DictConfig*) – DictConfig containing the selected project configuration.

Returns None.

Return type None

run()

Generate all files and directories registered inside the record instance.

Return type None

17.1.3 Package Contents

mloq.**_logger**

CHAPTER

EIGHTEEN

INDICES AND TABLES

genindex modindex search

PYTHON MODULE INDEX

m

INDEX

Symbols

A

B

C

D

E

F

G

H

I

J

L

M

N

O

P

R

S

T

U

V

W