whey

Release 0.1.0

A simple Python wheel builder for simple projects.

Dominic Davis-Foster

May 15, 2024

Contents

> Little Miss Muffet
> She sat on a tuffet,
> Eating of curds and whey;
> There came a little spider,
> Who sat down beside her,
> And frighten'd Miss Muffet away.

`whey`:

- supports **PEP 621** metadata.
- can be used as a **PEP 517** build backend.
- creates **PEP 427** wheels.
- handles type hint files (py.typed and `*.pyi` stubs).
- is distributed under the MIT License.
- is the liquid remaining after milk has been curdled and strained. It is a byproduct of the manufacture of cheese and has several commercial uses.

Fig. 1: Miss Muffet sitting on a tuffet, by John Everett Millais, 1884

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install whey --user
```

1.2 from Anaconda

First add the required channels

```
$ conda config --add channels https://conda.anaconda.org/conda-forge
```

Then install

```
$ conda install whey
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/repo-helper/whey@master --user
```

`whey` also has an optional README validation feature, which checks the README will render correctly on PyPI. This requires that the `readme` extra is installed:

```
$ python -m pip install whey[readme]
```

and in `pyproject.toml`:

```
[build-system]
requires = [ "whey[readme]",]
build-backend = "whey"
```

Once the dependencies are installed the validation can be disabled by setting the *CHECK_README* environment variable to `0`.

Part I

Documentation

Chapter

TWO

Configuration

`whey` is configured in the `pyproject.toml` file defined in **PEP 517** and **PEP 518**.

Note: `whey` only supports TOML v0.5.0. `pyproject.toml` files using features of newer TOML versions may not parse correctly.

2.1 [build-system]

`whey` must be set as the `build-backend` in the `[build-system]` table.

Example:

```toml
[build-system]
requires = [ "whey",]
build-backend = "whey"
```

2.2 [project]

The metadata used by `whey` is defined in the `[project]` table, per **PEP 621**.

As a minimum, the table MUST contain the keys *`name`* and *`version`*[1].

`name`

Type: String
Required: `True`

The name of the project.

Ideally, the name should be normalised to lowercase, with underscores replaced by hyphens. The name may only contain ASCII letters, numbers, and the following symbols: `._-` . It must start and end with a letter or number.

This key is required, and MUST be defined statically.

Example:

```toml
[project]
name = "spam"
```

[1] Other tools, such as flit and trampolim, may support determining *`project.version`* dynamically without specifying a value in `pyproject.toml`.

version

Type: String

The version of the project as supported by **PEP 440**.

With `whey` this key is required, and must be defined statically. Other backends may support determining this value automatically if it is listed in *`project.dynamic`*.

Example:

```
[project]
version = "2020.0.0"
```

description

Type: String

A short summary description of the project.

PyPI will display this towards the top of the project page. A longer description can be provided as *`readme`*.

Example:

```
[project]
description = "Lovely Spam! Wonderful Spam!"
```

readme

Type: String or table

The full description of the project (i.e. the README).

The field accepts either a string or a table. If it is a string then it is the relative path to a text file containing the full description. The file's encoding MUST be UTF-8, and have one of the following content types:

- `text/markdown`, with a a case-insensitive `.md` suffix.
- `text/x-rst`, with a a case-insensitive `.rst` suffix.
- `text/plain`, with a a case-insensitive `.txt` suffix.

The readme field may instead be a table with the following keys:

- `file` – a string value representing a relative path to a file containing the full description.
- `text` – a string value which is the full description.
- `content-type` – (required) a string specifying the content-type of the full description.
- `charset` – (optional, default UTF-8) the encoding of the `file`.

The `file` and `text` keys are mutually exclusive, but one must be provided in the table.

PyPI will display this on the project page

Examples:

```
[project]
readme = "README.rst"
```

```
[project]
readme = {file = "README.md", content-type = "text/markdown", encoding = "UTF-8"}
```

```toml
[project.readme]
text = "Spam is a brand of canned cooked pork made by Hormel Foods Corporation."
content-type = "text/x-rst"
```

requires-python

Type: String

The Python version requirements of the project, as a **PEP 508** specifier.

Example:

```toml
[project]
requires-python = ">=3.6"
```

license

Type: Table

The table may have one of two keys:

- `file` – a string value that is a relative file path to the file which contains the license for the project. The file's encoding MUST be UTF-8.
- `text` – string value which is the license of the project.

These keys are mutually exclusive.

Examples:

```toml
[project]
license = {file = "LICENSE.rst"}
```

```toml
[project.license]
file = "COPYING"
```

```toml
[project.license]
text = """
This software may only be obtained by sending the author a postcard,
and then the user promises not to redistribute it.
"""
```

authors

Type: Array of tables with string keys and values

The tables list the people or organizations considered to be the "authors" of the project.

Each table has 2 keys: `name` and `email`. Both keys are optional, and both values must be strings.

- The `name` value MUST be a valid email name (i.e. whatever can be put as a name, before an email, in **RFC 822**) and not contain commas.
- The `email` value MUST be a valid email address.

Examples:

```
[project]
authors = [
    {name = "Dominic Davis-Foster", email = "dominic@davis-foster.co.uk"},
    {name = "The pip developers", email = "distutils-sig@python.org"}
]
```

```
[[project.authors]]
name = "Tzu-ping Chung"

[[project.authors]]
email = "hi@pradyunsg.me"
```

`maintainers`

Type: Array of tables with string keys and values

The tables list the people or organizations considered to be the "maintainers" of the project.

This field otherwise functions the same as *`authors`*.

Example:

```
[project]
authors = [
  {email = "hi@pradyunsg.me"},
  {name = "Tzu-ping Chung"}
]
maintainers = [
  {name = "Brett Cannon", email = "brett@python.org"}
]
```

`keywords`

Type: Array of strings

The keywords for the project.

These can be used by community members to find projects based on their desired criteria.

Example:

```
[project]
keywords = [ "egg", "bacon", "sausage", "tomatoes", "Lobster Thermidor",]
```

`classifiers`

Type: Array of strings

The trove classifiers which apply to the project.

Classifiers describe who the project is for, what systems it can run on, and how mature it is. These can then be used by community members to find projects based on their desired criteria.

Example:

```
[project]
classifiers = [
    "Development Status :: 4 - Beta",
    "Programming Language :: Python"
]
```

urls

Type: Table, with keys and values of strings

A table of URLs where the key is the URL label and the value is the URL itself.

The URL labels are free text, but may not exceed 32 characters.

Example:

```
[project.urls]
homepage = "https://example.com"
documentation = "https://readthedocs.org"
repository = "https://github.com"
changelog = "https://github.com/me/spam/blob/master/CHANGELOG.md"
```

scripts

Type: Table, with keys and values of strings

The console scripts provided by the project.

The keys are the names of the scripts and the values are the object references in the form `module.submodule:object`.

See the entry point specification for more details.

Example:

```
[project.scripts]
spam-cli = "spam:main_cli"
# One which depends on extras:
foobar = "foomod:main_bar [bar,baz]"
```

gui-scripts

Type: Table, with keys and values of strings

The graphical application scripts provided by the project.

The keys are the names of the scripts, and the values are the object references in the form `module.submodule:object`.

See the entry point specification for more details.

Example:

```
[project.gui-scripts]
spam-gui = "spam.gui:main_gui"
```

`entry-points`

Type: Table of tables, with keys and values of strings

Each sub-table's name is an entry point group.

Users MUST NOT create nested sub-tables but instead keep the entry point groups to only one level deep.

Users MUST NOT create sub-tables for `console_scripts` or `gui_scripts`. Use `[project.scripts]` and `[project.gui-scripts]` instead.

See the entry point specification for more details.

Example:

```
[project.entry-points."spam.magical"]
tomatoes = "spam:main_tomatoes"

# pytest plugins refer to a module, so there is no ':obj'
[project.entry-points.pytest11]
nbval = "nbval.plugin"
```

`dependencies`

Type: Array of **PEP 508** strings

The dependencies of the project.

Each string MUST be formatted as a valid **PEP 508** string.

Example:

```
[project]
dependencies = [
    "httpx",
    "gidgethub[httpx]>4.0.0",
    "django>2.1; os_name != 'nt'",
    "django>2.0; os_name == 'nt'"
]
```

`optional-dependencies`

Type: Table with values of arrays of **PEP 508** strings

The optional dependencies of the project.

- The keys specify an extra, and must be valid Python identifiers.
- The values are arrays of strings, which must be valid **PEP 508** strings.

Example:

```
[project.optional-dependencies]
test = [
  "pytest < 5.0.0",
  "pytest-cov[all]"
]
```

dynamic

Type: Array of strings

Specifies which fields listed by **PEP 621** were intentionally unspecified so `whey` can provide such metadata dynamically.

Whey currently only supports *`classifiers`*, *`dependencies`*, and *`requires-python`* as dynamic fields. Other tools may support different dynamic fields.

Example:

```
[project]
dynamic = [ "classifiers",]

[tool.whey]
base-classifiers = [
    "Development Status :: 3 - Alpha",
    "Typing :: Typed",
]
```

2.3 [tool.whey]

package

Type: String

The path to the package to distribute, relative to the directory containing `pyproject.toml`. This defaults to *`project.name`* if unspecified.

Example:

```
[project]
name = "domdf-python-tools"

[tool.whey]
package = "domdf_python_tools"
```

source-dir

Type: String

The name of the directory containing the project's source. This defaults to `'.'` if unspecified.

Examples:

```
[project]
name = "flake8"
```

```
[tool.whey]
source-dir = "src/flake8"
```

additional-files

Type: Array of strings

A list of MANIFEST.in-style entries for additional files to include in distributions.

The supported commands are:

Command	Description
`include pat1 pat2 ...`	Add all files matching any of the listed patterns
`exclude pat1 pat2 ...`	Remove all files matching any of the listed patterns
`recursive-include dir-pattern pat1 pat2 ...`	Add all files under directories matching `dir-pattern` that match any of the listed patterns
`recursive-exclude dir-pattern pat1 pat2 ...`	Remove all files under directories matching `dir-pattern` that match any of the listed patterns

`whey` was built with type hints in mind, so it will automatically include any `py.typed` files and `*.pyi` stub files automatically.

Note: If using *`tool.whey.source-dir`*, the entries for files within the package must start with the value of *`source-dir`*.

For example, if *`source-dir`* is `'src'` and the package is at `src/spam` an entry might be `include src/spam/template.scss`.

Examples:

```toml
[tool.whey]
additional-files = [
    "include domdf_python_tools/google-10000-english-no-swears.txt",
    "recursive-exclude domdf_python_tools *.json",
]
```

```toml
[tool.whey]
source-dir = "src"
additional-files = [
    "include src/domdf_python_tools/google-10000-english-no-swears.txt",
    "recursive-exclude src/domdf_python_tools *.json",
]
```

license-key

Type: String

An identifier giving the project's license.

This is used for the License field in the Core Metadata, and to add the appropriate trove classifier.

It is recommended to use an SPDX Identifier, but note that not all map to classifiers.

Example:

```toml
[tool.whey]
license-key = "MIT"
```

base-classifiers

Type: Array of strings

A list of trove classifiers.

This list will be extended with the appropriate classifiers for the *license-key* and the supported *platforms*, *python-implementations* and *python-versions*.

This field is ignored if *classifiers* is not listed in *project.dynamic*

Example:

```
[project]
dynamic = [ "classifiers", ]

[tool.whey]
base-classifiers = [
    "Development Status :: 3 - Alpha",
    "Typing :: Typed",
]
```

platforms

Type: Array of strings

A list of supported platforms. This is used to add appropriate trove classifiers and is listed under Platform in the Core Metadata.

Example:

```
[tool.whey]
platforms = [ "Windows", "Linux",]
```

python-implementations

Type: Array of strings

A list of supported Python implementations. This can be used to add appropriate trove classifiers.

Example:

```
[tool.whey]
python-implementations = [ "CPython", "PyPy",]
```

python-versions

Type: Array of strings

A list of supported Python versions. This can be used to add appropriate trove classifiers and dynamically determine the minimum required Python version for *project.requires-python*.

Example:

```
[tool.whey]
python-versions = [
    "3.6",
    "3.7",
]
```

2.4 Environment Variables

CHECK_README
: Setting this to `0` disables the optional README validation feature, which checks the README will render correctly on PyPI.

SOURCE_DATE_EPOCH
: To make reproducible builds, set this to a timestamp as a number of seconds since 1970-01-01 UTC, and document the value you used. On Unix systems, you can get a value for the current time by running:

```
$ date +%s
```

Note: The timestamp cannot be before 1980-01-01 or after 2107-12-31.

See also:

The SOURCE_DATE_EPOCH specification

WHEY_VERBOSE
: Run whey in verbose mode. This includes printing the names of the files being added to the sdist or wheel.

If using whey's command-line interface, this option defaults to `0`. Setting it to `1` has the same meaning as the *`-v / --verbose`* option.

If using whey's **PEP 517** backend, this option defaults to `1`. Setting it to `0` disables the verbose output.

WHEY_TRACEBACK
: Show the complete traceback on error.

This option defaults to `0`. Setting it to `1` has the same meaning as the *`-T / --traceback`* option, both for the command-line interface and the **PEP 517** backend.

2.5 Complete Example

This is an example of a complete `pyproject.toml` file for **PEP 621**.

For an explanation of each field, see the *Configuration* section.

Listing 1: `pyproject.toml`

```toml
[build-system]
requires = [ "whey",]
build-backend = "whey"

[project]
name = "spam"
version = "2020.0.0"
description = "Lovely Spam! Wonderful Spam!"
readme = "README.rst"
requires-python = ">=3.8"
license = {file = "LICENSE.txt"}
keywords = [ "egg", "bacon", "sausage", "tomatoes", "Lobster Thermidor",]
```

(continues on next page)

(continued from previous page)

```
authors = [
    {name = "Dominic Davis-Foster", email = "dominic@davis-foster.co.uk"},
    {name = "The pip developers", email = "distutils-sig@python.org"}
]
maintainers = [
    {name = "Brett Cannon", email = "brett@python.org"}
]
classifiers = [
    "Development Status :: 4 - Beta",
    "Programming Language :: Python"
]

dependencies = [
    "httpx",
    "gidgethub[httpx]>4.0.0",
    "django>2.1; os_name != 'nt'",
    "django>2.0; os_name == 'nt'"
]

[project.optional-dependencies]
test = [
    "pytest < 5.0.0",
    "pytest-cov[all]"
]

[project.urls]
homepage = "https://example.com"
documentation = "https://readthedocs.org"
repository = "https://github.com"
changelog = "https://github.com/me/spam/blob/master/CHANGELOG.md"

[project.scripts]
spam-cli = "spam:main_cli"
# One which depends on extras:
foobar = "foomod:main_bar [bar,baz]"

[project.gui-scripts]
spam-gui = "spam:main_gui"

[project.entry-points."spam.magical"]
tomatoes = "spam:main_tomatoes"

# pytest plugins refer to a module, so there is no ':obj'
[project.entry-points.pytest11]
nbval = "nbval.plugin"
```

Chapter

THREE

Command Line Usage

3.1 whey

Build a wheel for the given project.

```
whey [OPTIONS] [PROJECT]
```

Options

-s, --sdist
Build a source distribution.

-w, --wheel
Build a wheel.

-b, --binary
Build a binary distribution.

-B, --builder <BUILDER>
The builder to build with.

--build-dir <DIRECTORY>
The temporary build directory.

-o, --out-dir <DIRECTORY>
The output directory.

-v, --verbose
Enable verbose output.

-S, --show-builders
Show the builders which will be used, and exit.

--colour, --no-colour
Whether to use coloured output.

-T, --traceback
Show the complete traceback on error.

--version
Show the version and exit.

Arguments

PROJECT
The path to the project to build.

Optional argument. Default `'.'`

Environment variables

WHEY_VERBOSE
Provides a default for *-v / --verbose*

WHEY_TRACEBACK
Provides a default for *-T / --traceback*

3.2 Editable installs

Whey also supports **PEP 660** editable installs via pip. Editable installs allow changes to the project's source code (but not its entry points and other metadata) to be automatically reflected when the module is next imported.

To install the project in the current directory in editable mode, run the following command:

```
$ python3 -m pip install --editable .
```

See the pip documentation for more details.

If using pip's `--no-build-isolation` flag[1], whey must be installed with the `editable` extra, as additional requirements are required for editable installs.

[1] https://pip.pypa.io/en/stable/cli/pip_install/#cmdoption-no-build-isolation

Chapter

FOUR

Downloading source code

The `whey` source code is available on GitHub, and can be accessed from the following URL: `https://github.com/repo-helper/whey`

If you have `git` installed, you can clone the repository with the following command:

```
$ git clone https://github.com/repo-helper/whey
```

```
Cloning into 'whey'...
remote: Enumerating objects: 47, done.
remote: Counting objects: 100% (47/47), done.
remote: Compressing objects: 100% (41/41), done.
remote: Total 173 (delta 16), reused 17 (delta 6), pack-reused 126
Receiving objects: 100% (173/173), 126.56 KiB | 678.00 KiB/s, done.
Resolving deltas: 100% (66/66), done.
```

Alternatively, the code can be downloaded in a 'zip' file by clicking:
Clone or download –> Download Zip

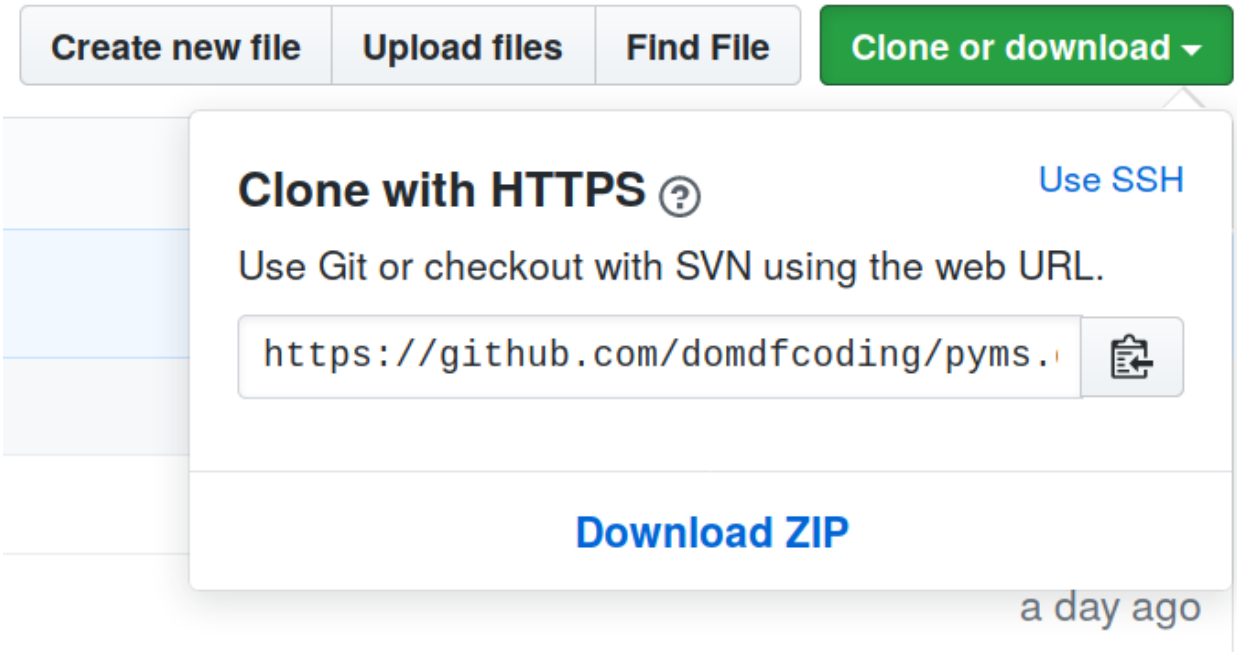

Fig. 1: Downloading a 'zip' file of the source code

4.1 Building from source

The recommended way to build `whey` is to use tox:

```
$ tox -e build
```

The source and wheel distributions will be in the directory `dist`.

If you wish, you may also use pep517.build or another **PEP 517**-compatible build tool.

Chapter

FIVE

License

`whey` is licensed under the MIT License

A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.

Permissions

- Commercial use – The licensed material and derivatives may be used for commercial purposes.
- Modification – The licensed material may be modified.
- Distribution – The licensed material may be distributed.
- Private use – The licensed material may be used and modified in private.

Conditions

- License and copyright notice – A copy of the license and copyright notice must be included with the licensed material.

Limitations

- Liability – This license includes a limitation of liability.
- Warranty – This license explicitly states that it does NOT provide any warranty.

See more information on choosealicense.com ⇒

```
Copyright (c) 2021 Dominic Davis-Foster

Permission is hereby granted, free of charge, to any person obtaining a copy
of this software and associated documentation files (the "Software"), to deal
in the Software without restriction, including without limitation the rights
to use, copy, modify, merge, publish, distribute, sublicense, and/or sell
copies of the Software, and to permit persons to whom the Software is
furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all
copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND,
EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.
```

Part II

API Reference

Chapter

SIX

whey.additional_files

Parser for the `additional-files` option.

Classes:

AdditionalFilesEntry()	An abstract command in `additional-files`.
Exclude(patterns)	Exclude a single file, or multiple files with a pattern.
Include(patterns)	Include a single file, or multiple files with a pattern.
RecursiveExclude(path, patterns)	Recursively exclude files in a directory based on patterns.
RecursiveInclude(path, patterns)	Recursively include files in a directory based on patterns.

Functions:

from_entry(line)	Parse a MANIFEST.in-style entry.

class AdditionalFilesEntry

Bases: ABC

An abstract command in `additional-files`.

Methods:

iter_files(directory)	Returns an iterator over files to be included or excluded by this command.
parse(parameters)	Parse the command's parameters.
to_dict()	Returns a dictionary representation of the command entry.

abstract iter_files(*directory*)

Returns an iterator over files to be included or excluded by this command.

Parameters **directory** (PathPlus) – The project or build directory.

Return type Iterator[PathPlus]

abstract classmethod parse(*parameters*)

Parse the command's parameters.

Parameters **parameters** (str)

Return type *AdditionalFilesEntry*

abstract to_dict()

Returns a dictionary representation of the command entry.

Return type Dict[str, Any]

class Exclude(*patterns*)

Bases: *whey.additional_files.AdditionalFilesEntry*

Exclude a single file, or multiple files with a pattern.

Parameters **patterns** (Iterable[str]) – Glob patterns.

Methods:

iter_files(directory)	Returns an iterator over files to be excluded by this command.
parse(parameters)	Parse the command's parameters.
to_dict()	Returns a dictionary representation of the command entry.

Attributes:

patterns	Glob patterns (with complete paths from the project root)

iter_files(*directory*)

Returns an iterator over files to be excluded by this command.

Parameters **directory** (PathPlus) – The build directory.

Return type Iterator[PathPlus]

classmethod parse(*parameters*)

Parse the command's parameters.

Parameters **parameters** (str)

Return type *Exclude*

patterns

Type: List[str]

Glob patterns (with complete paths from the project root)

to_dict()

Returns a dictionary representation of the command entry.

Return type Dict[str, Any]

class Include(*patterns*)

Bases: *whey.additional_files.AdditionalFilesEntry*

Include a single file, or multiple files with a pattern.

Parameters **patterns** (Iterable[str]) – Glob patterns.

Methods:

iter_files(directory)	Returns an iterator over files to be included by this command.
parse(parameters)	Parse the command's parameters.
to_dict()	Returns a dictionary representation of the command entry.

Attributes:

patterns	Glob patterns (with complete paths from the project root)

iter_files(*directory*)
Returns an iterator over files to be included by this command.

Parameters **directory** (PathPlus) – The project directory.

Return type Iterator[PathPlus]

classmethod parse(*parameters*)
Parse the command's parameters.

Parameters **parameters** (str)

Return type *Include*

patterns
Type: List[str]

Glob patterns (with complete paths from the project root)

to_dict()
Returns a dictionary representation of the command entry.

Return type Dict[str, Any]

class RecursiveExclude(*path*, *patterns*)
Bases: *whey.additional_files.AdditionalFilesEntry*

Recursively exclude files in a directory based on patterns.

Parameters

- **path** (str) – The directory to start from.
- **patterns** (Iterable[str]) – Glob patterns.

Methods:

iter_files(directory)	Returns an iterator over files to be excluded by this command.
parse(parameters)	Parse the command's parameters.
to_dict()	Returns a dictionary representation of the command entry.

Attributes:

path	The directory to start from.
patterns	Glob patterns.

iter_files(*directory*)
Returns an iterator over files to be excluded by this command.

Parameters **directory** (PathPlus) – The build directory.

Return type Iterator[PathPlus]

classmethod parse(*parameters*)
Parse the command's parameters.

Parameters **parameters** (str)

Return type *RecursiveExclude*

path
Type: str

The directory to start from.

patterns
Type: List[str]

Glob patterns.

to_dict()
Returns a dictionary representation of the command entry.

Return type Dict[str, Any]

class RecursiveInclude(*path*, *patterns*)
Bases: *whey.additional_files.AdditionalFilesEntry*

Recursively include files in a directory based on patterns.

Parameters

- **path** (str) – The directory to start from.
- **patterns** (Iterable[str]) – Glob patterns.

Methods:

iter_files(directory)	Returns an iterator over files to be included by this command.
parse(parameters)	Parse the command's parameters.
to_dict()	Returns a dictionary representation of the command entry.

Attributes:

path	The directory to start from.
patterns	Glob patterns.

iter_files(*directory*)
Returns an iterator over files to be included by this command.

Parameters **directory** (PathPlus) – The project directory.

Return type Iterator[PathPlus]

classmethod parse(*parameters*)
Parse the command's parameters.

Parameters **parameters** (str)

Return type *RecursiveInclude*

path
Type: str

The directory to start from.

patterns
Type: List[str]

Glob patterns.

to_dict()
Returns a dictionary representation of the command entry.

Return type Dict[str, Any]

from_entry(*line*)
Parse a MANIFEST.in-style entry.

Parameters **line** (str)

Return type Optional[*AdditionalFilesEntry*]

Returns An *AdditionalFilesEntry* for known commands, or None if an unknown command is found in the entry.

Chapter

SEVEN

whey.builder

The actual wheel builder.

Classes:

AbstractBuilder(project_dir, config[, ...])	Abstract base class for builders of Python distributions using metadata read from `pyproject.toml`.
SDistBuilder(project_dir, config[, ...])	Builds source distributions using metadata read from `pyproject.toml`.
WheelBuilder(project_dir, config[, ...])	Builds wheel binary distributions using metadata read from `pyproject.toml`.

class AbstractBuilder(*project_dir*, *config*, *build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Bases: ABC

Abstract base class for builders of Python distributions using metadata read from `pyproject.toml`.

Parameters

- **project_dir** (PathPlus) – The project to build the distribution for.
- **build_dir** (Union[str, Path, PathLike, None]) – The (temporary) build directory. Default *<project_dir>*/build/.
- **out_dir** (Union[str, Path, PathLike, None]) – The output directory. Default *<project_dir>*/dist/.
- **verbose** (bool) – Whether to enable verbose output. Default False.
- **colour** (Optional[bool]) – Whether to use coloured output. Default None.

Attributes:

archive_name	The archive name, without the tag
build_dir	The (temporary) build directory.
code_directory	The directory containing the code in the build directory.
colour	Whether to use coloured output.
config	Configuration parsed from `pyproject.toml`.
default_build_dir	Provides a default for the `build_dir` argument.
default_out_dir	Provides a default for the `out_dir` argument.
out_dir	The output directory.
project_dir	The pyproject.toml directory
verbose	Whether to enable verbose output.

Methods:

build()	Build the distribution.
call_additional_hooks()	Subclasses may call this method to give *their* subclasses an opportunity to run custom code.
clear_build_dir()	Clear the build directory of any residue from previous builds.
copy_additional_files()	Copy additional files to the build directory, as specified in the `additional-files` key.
copy_source()	Copy source files into the build directory.
get_metadata_map()	Generate the content of the `METADATA` / `PKG-INFO` file.
iter_source_files()	Iterate over the files in the source directory.
parse_additional_files(*entries)	Copy additional files to the build directory, by parsing MANIFEST.in-style entries.
parse_authors()	Parse the *project.authors* and *maintainers* fields into Author, Maintainer-Email etc.
report_copied(source, target)	Report that a file has been copied into the build directory.
report_removed(removed_file)	Reports the removal of a file from the build directory.
report_written(written_file)	Report that a file has been written to the build directory.
write_license(dest_dir[, dest_filename])	Write the `LICENSE` file.
write_metadata(metadata_file, metadata_mapping)	Write Core Metadata to the given file.

archive_name
The archive name, without the tag

abstract build()
Build the distribution.

Return type str

Returns The filename of the created archive.

build_dir
The (temporary) build directory.

call_additional_hooks()
Subclasses may call this method to give *their* subclasses an opportunity to run custom code.

For example, the wheel builder calls this as the final step before adding files to the archive, giving an opportunity for subclasses of *WheelBuilder* to include additional steps without having to override the entire *build_wheel()* method.

clear_build_dir()
Clear the build directory of any residue from previous builds.

property code_directory
The directory containing the code in the build directory.

Return type str

colour
Whether to use coloured output.

config
Type: Dict[str, Any]

Configuration parsed from pyproject.toml.

copy_additional_files()
Copy additional files to the build directory, as specified in the additional-files key.

copy_source()
Copy source files into the build directory.

property default_build_dir
Provides a default for the build_dir argument.

Return type PathPlus

property default_out_dir
Provides a default for the out_dir argument.

Return type PathPlus

get_metadata_map()
Generate the content of the METADATA / PKG-INFO file.

Return type MetadataMapping

iter_source_files()
Iterate over the files in the source directory.

Return type Iterator[PathPlus]

out_dir
The output directory.

parse_additional_files(**entries*)
Copy additional files to the build directory, by parsing MANIFEST.in-style entries.

Parameters ***entries** (*AdditionalFilesEntry*)

parse_authors()
Parse the *project.authors* and *maintainers* fields into Author, Maintainer-Email etc.

Return type Dict[str, str]

Returns

A mapping of field names to values.

Possible field names are Author, Author-Email, Maintainer, and Maintainer-Email.

project_dir
Type: PathPlus

The pyproject.toml directory

report_copied(*source*, *target*)
Report that a file has been copied into the build directory.

The format is:

```
Copying {source} -> {target.relative_to(self.build_dir)}
```

Parameters

- **source** (Path) – The source file
- **target** (Path) – The file in the build directory.

report_removed(*removed_file*)
Reports the removal of a file from the build directory.

The format is:

```
Removing {removed_file.relative_to(self.build_dir)}
```

Parameters **removed_file** (Path)

report_written(*written_file*)
Report that a file has been written to the build directory.

The format is:

```
Writing {written_file.relative_to(self.build_dir)}
```

Parameters **written_file** (Path)

verbose
Whether to enable verbose output.

write_license(*dest_dir*, *dest_filename='LICENSE'*)
Write the LICENSE file.

Parameters

- **dest_dir** (PathPlus) – The directory to write the file into.
- **dest_filename** (str) – The name of the file to write in dest_dir. Default 'LICENSE'.

write_metadata(*metadata_file*, *metadata_mapping*)
Write Core Metadata to the given file.

Parameters **metadata_file** (PathPlus)

class SDistBuilder(*project_dir*, *config*, *build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Bases: *AbstractBuilder*

Builds source distributions using metadata read from `pyproject.toml`.

Parameters

- **project_dir** (PathPlus) – The project to build the distribution for.
- **build_dir** (Union[str, Path, PathLike, None]) – The (temporary) build directory. Default *<project_dir>*/build/sdist.
- **out_dir** (Union[str, Path, PathLike, None]) – The output directory. Default *<project_dir>*/dist.
- **verbose** (bool) – Enable verbose output. Default False.

Methods:

build()	Build the source distribution.
build_sdist()	Build the source distribution.
create_sdist_archive()	Create the sdist archive.
write_pyproject_toml()	Write the `pyproject.toml` file.
write_readme()	Write the `README.*` file.

Attributes:

code_directory	The directory containing the code in the build and project directories.
default_build_dir	Provides a default for the `build_dir` argument.

build()

Build the source distribution.

Return type str

Returns The filename of the created archive.

build_sdist()

Build the source distribution.

Return type str

Returns The filename of the created archive.

property code_directory

The directory containing the code in the build and project directories.

Return type str

create_sdist_archive()

Create the sdist archive.

Return type str

Returns The filename of the created archive.

property default_build_dir
Provides a default for the `build_dir` argument.

Return type PathPlus

write_pyproject_toml()
Write the `pyproject.toml` file.

write_readme()
Write the `README.*` file.

class WheelBuilder(*project_dir*, *config*, *build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Bases: *AbstractBuilder*

Builds wheel binary distributions using metadata read from `pyproject.toml`.

Parameters

- **project_dir** (PathPlus) – The project to build the distribution for.
- **build_dir** (Union[str, Path, PathLike, None]) – The (temporary) build directory. Default *<project_dir>*/build/wheel.
- **out_dir** (Union[str, Path, PathLike, None]) – The output directory. Default *<project_dir>*/dist.
- **verbose** (bool) – Enable verbose output. Default False.

Methods:

build()	Build the binary wheel distribution.
build_editable()	Build an editable wheel.
build_wheel()	Build the binary wheel distribution.
create_editables_files()	Generate files with editables for use in a **PEP 660** wheel.
create_wheel_archive()	Create the wheel archive.
get_source_epoch()	Returns the parsed value of the *SOURCE_DATE_EPOCH* environment variable, or None if unset.
write_entry_points()	Write the list of entry points to the wheel, as specified in [project.scripts], [project.gui-scripts] and [project.entry-points]
write_wheel()	Write the metadata to the WHEEL file.

Attributes:

default_build_dir	Provides a default for the build_dir argument.
dist_info	The *.dist-info directory in the build directory.
generator	The value for the Generator field in *.dist-info/WHEEL.
tag	The tag for the wheel.

build()

Build the binary wheel distribution.

Return type str

Returns The filename of the created archive.

build_editable()

Build an editable wheel.

An "editable" wheel uses the wheel format not for distribution but as ephemeral communication between the build system and the front end. This avoids having the build backend install anything directly. This wheel must not be exposed to end users, nor cached, nor distributed.

You should use a different build_dir and out_dir to those used for standard wheel builds.

The default implementation of this method does not call *copy_source()* or *copy_additional_files()*.

Attention: This method has the following additional requirement:

```
editables>=0.2
```

This can be installed as follows:

```
$ python -m pip install whey[editable]
```

Return type str

Returns The filename of the created archive.

build_wheel()
Build the binary wheel distribution.

Return type str

Returns The filename of the created archive.

create_editables_files()
Generate files with editables for use in a **PEP 660** wheel.

Attention: This method has the following additional requirement:

```
editables>=0.2
```

This can be installed as follows:

```
$ python -m pip install whey[editable]
```

Return type Iterator[ComparableRequirement]

Returns An iterator of additional runtime requirements which should be added to the wheel's METADATA file.

create_wheel_archive()
Create the wheel archive.

Return type str

Returns The filename of the created archive.

property default_build_dir
Provides a default for the build_dir argument.

Return type PathPlus

property dist_info
The *.dist-info directory in the build directory.

Return type PathPlus

property generator
The value for the `Generator` field in `*.dist-info/WHEEL`.

Return type `str`

static get_source_epoch()
Returns the parsed value of the *SOURCE_DATE_EPOCH* environment variable, or `None` if unset.

See `https://reproducible-builds.org/specs/source-date-epoch/` for the specification.

Raises **ValueError** – if the value is in an invalid format.

Return type `Optional[datetime]`

property tag
The tag for the wheel.

Return type `str`

write_entry_points()
Write the list of entry points to the wheel, as specified in `[project.scripts]`, `[project.gui-scripts]` and `[project.entry-points]`

write_wheel()
Write the metadata to the `WHEEL` file.

Chapter

EIGHT

whey.config

pyproject.toml configuration parsing.

Functions:

load_toml(filename)	Load the whey configuration mapping from the given TOML file.

load_toml (*filename*)
Load the whey configuration mapping from the given TOML file.

Parameters **filename** (Union[str, Path, PathLike])

Return type Dict[str, Any]

8.2 whey.config.pep621

PEP 621 configuration parser.

Classes:

PEP621Parser()	Parser for PEP 621 metadata from pyproject.toml.

class PEP621Parser
Bases: PEP621Parser

Parser for PEP 621 metadata from pyproject.toml.

Methods:

parse(config[, set_defaults])	Parse the TOML configuration.

parse (*config*, *set_defaults=False*)
Parse the TOML configuration.

Parameters

- **config** (Dict[str, Any])
- **set_defaults** (bool) – If True, the values in dom_toml.parser.AbstractConfigParser.defaults and dom_toml.parser.AbstractConfigParser.factories will be set as defaults for the returned mapping. Default False.

Return type ProjectDict

8.3 whey.config.whey

Parser for whey's own configuration.

Classes:

WheyParser()	Parser for the `[tool.whey]` table from `pyproject.toml`.

Functions:

backfill_classifiers(config)	Backfill trove classifiers for supported platforms, Python versions and implementations, and the project's license, as appropriate.
get_default_builders()	Returns a mapping of builder categories to builder classes to use as the default builders.
get_entry_points([group])	Returns an iterable over EntryPoint objects in the `group` group.

Data:

license_lookup	Mapping of license short codes to license names used in trove classifiers.

class WheyParser

Bases: AbstractConfigParser

Parser for the `[tool.whey]` table from `pyproject.toml`.

property keys

The keys to parse from the TOML file.

Return type List[str]

parse_additional_files(*config*)

Parse the `additional-files` key, giving MANIFEST.in-style entries for additional files to include in distributions.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type List[*AdditionalFilesEntry*]

parse_base_classifiers(*config*)

Parse the `base-classifiers` key, giving a list trove classifiers.

This list will be extended with the appropriate classifiers for supported platforms, Python versions and implementations, and the project's license. Ignored if classifiers is not listed in dynamic

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type Set[str]

parse_builders(*config*)
Parse the `builders` table, which lists gives the entry points to use for the sdist and wheel builders.

This allows the user to select a custom builder with additional functionality.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type Dict[str, Type[*AbstractBuilder*]]

parse_license_key(*config*)
Parse the `license-key` key, giving the identifier of the project's license. Optional.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type str

parse_package(*config*)
Parse the `package` key, giving the name of the importable package.

This defaults to project.name if unspecified.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type str

parse_platforms(*config*)
Parse the `platforms` key, giving a list of supported platforms. Optional.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type List[str]

parse_python_implementations(*config*)
Parse the `python-implementations` key, giving a list of supported Python implementations. Optional.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type List[str]

static parse_python_versions(*config*)
Parse the `python-versions` key, giving a list of supported Python versions. Optional.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type List[str]

parse_source_dir(*config*)
Parse the `source-dir` key, giving the name of the directory containing the project's source.

This defaults to `'.'` if unspecified.

Parameters **config** (Dict[str, Any]) – The unparsed TOML config for the `[tool.whey]` table.

Return type str

backfill_classifiers(*config*)
Backfill trove classifiers for supported platforms, Python versions and implementations, and the project's license, as appropriate.

Parameters **config** (Dict[str, Any]) – The parsed config from `pyproject.toml`.

Return type List[str]

get_default_builders()
Returns a mapping of builder categories to builder classes to use as the default builders.

Return type Dict[str, Type[*AbstractBuilder*]]

get_entry_points(*group='whey.builder'*)
Returns an iterable over EntryPoint objects in the `group` group.

Parameters **group** (str) – Default `'whey.builder'`.

Return type Iterable[EntryPoint]

license_lookup
Type: dict

Mapping of license short codes to license names used in trove classifiers.

```
{
  "Apache-2.0": "Apache Software License",
  "BSD": "BSD License",
  "BSD-2-Clause": "BSD License",
  "BSD-3-Clause": "BSD License",
  "AGPL-3.0-only": "GNU Affero General Public License v3",
  "AGPL-3.0": "GNU Affero General Public License v3",
  "AGPL-3.0-or-later": "GNU Affero General Public License v3 or later (AGPLv3+)",
  "AGPL-3.0+": "GNU Affero General Public License v3 or later (AGPLv3+)",
  "FDL": "GNU Free Documentation License (FDL)",
  "GFDL-1.1-only": "GNU Free Documentation License (FDL)",
  "GFDL-1.1-or-later": "GNU Free Documentation License (FDL)",
  "GFDL-1.2-only": "GNU Free Documentation License (FDL)",
  "GFDL-1.2-or-later": "GNU Free Documentation License (FDL)",
  "GFDL-1.3-only": "GNU Free Documentation License (FDL)",
  "GFDL-1.3-or-later": "GNU Free Documentation License (FDL)",
  "GFDL-1.2": "GNU Free Documentation License (FDL)",
  "GFDL-1.1": "GNU Free Documentation License (FDL)",
  "GFDL-1.3": "GNU Free Documentation License (FDL)",
  "GPL": "GNU General Public License (GPL)",
  "GPL-1.0-only": "GNU General Public License (GPL)",
  "GPL-1.0-or-later": "GNU General Public License (GPL)",
  "GPLv2": "GNU General Public License v2 (GPLv2)",
  "GPL-2.0-only": "GNU General Public License v2 (GPLv2)",
  "GPLv2+": "GNU General Public License v2 or later (GPLv2+)",
  "GPL-2.0-or-later": "GNU General Public License v2 or later (GPLv2+)",
  "GPLv3": "GNU General Public License v3 (GPLv3)",
  "GPL-3.0-only": "GNU General Public License v3 (GPLv3)",
  "GPLv3+": "GNU General Public License v3 or later (GPLv3+)",
  "GPL-3.0-or-later": "GNU General Public License v3 or later (GPLv3+)",
  "LGPLv2": "GNU Lesser General Public License v2 (LGPLv2)",
  "LGPLv2+": "GNU Lesser General Public License v2 or later (LGPLv2+)",
  "LGPLv3": "GNU Lesser General Public License v3 (LGPLv3)",
  "LGPL-3.0-only": "GNU Lesser General Public License v3 (LGPLv3)",
  "LGPLv3+": "GNU Lesser General Public License v3 or later (LGPLv3+)",
  "LGPL-3.0-or-later": "GNU Lesser General Public License v3 or later (LGPLv3+)",
  "LGPL": "GNU Library or Lesser General Public License (LGPL)",
```

(continues on next page)

(continued from previous page)

```
  "MIT": "MIT License",
  "PSF-2.0": "Python Software Foundation License"
}
```

Chapter

NINE

whey.foreman

The foreman is responsible for loading the configuration calling the builders.

Classes:

Foreman(project_dir)	Responsible for loading the configuration calling the builders.

class Foreman (*project_dir*)

Bases: object

Responsible for loading the configuration calling the builders.

Methods:

build_binary([build_dir, out_dir, verbose, ...])	Build a binary distribution using the `binary` builder configured in `pyproject.toml`.
build_sdist([build_dir, out_dir, verbose, ...])	Build a sdist distribution using the `sdist` builder configured in `pyproject.toml`.
build_wheel([build_dir, out_dir, verbose, ...])	Build a wheel distribution using the `wheel` builder configured in `pyproject.toml`.
get_builder(distribution_type)	Returns the builder for the given distribution type.

Attributes:

config	Configuration parsed from `pyproject.toml`.
project_dir	The pyproject.toml directory

build_binary (*build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Build a binary distribution using the `binary` builder configured in `pyproject.toml`.

Return type str

Returns The filename of the created archive.

build_sdist (*build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Build a sdist distribution using the `sdist` builder configured in `pyproject.toml`.

Return type str

Returns The filename of the created archive.

build_wheel (*build_dir=None*, *out_dir=None*, **args*, *verbose=False*, *colour=None*, ***kwargs*)

Build a wheel distribution using the `wheel` builder configured in `pyproject.toml`.

Return type str

Returns The filename of the created archive.

config
Configuration parsed from pyproject.toml.

get_builder(*distribution_type*)
Returns the builder for the given distribution type.

Parameters **distribution_type** (str) – The distribution type, such as 'source' or 'wheel'.

Return type Type[*AbstractBuilder*]

project_dir
Type: PathPlus

The pyproject.toml directory

Chapter

TEN

Extending whey

`whey` can be extended to support building different distribution types (e.g. conda, DEB, RPM) or to modify the behaviour of an existing builder.

Custom builders must be registered as an entry point in the `whey.builder` group. For example:

```toml
# pyproject.toml

[project.entry-points."whey.builder"]
whey_sdist = "whey.builder:SDistBuilder"
whey_wheel = "whey.builder:WheelBuilder"
```

```ini
# setup.cfg

[options.entry_points]
whey.builder =
    whey_sdist = whey.builder:SDistBuilder
    whey_wheel = whey.builder:WheelBuilder
```

Each builder must inherit from *whey.builder.AbstractBuilder*.

The custom builders can be enabled by setting keys in the `tool.whey.builders` table. The table supports three keys: `sdist`, `wheel`, `binary`.

- The `sdist` builder is used when running whey with the *--sdist* option or when using the **PEP 517** backend to build an sdist.
- The `wheel` builder is used when running whey with the *--wheel* option or when using the **PEP 517** backend to build a wheel.
- The `binary` builder is used when running whey with the *--binary* option.

The value for each key is the name of an entry point, such as `whey_sdist` from the example above.

Python Module Index

W

Index

Symbols

A

B

C

D

E

F

G

I

K

L

M

O

P

R

S

T

V

W