dist-meta

Release 0.8.0

Parse and create Python distribution metadata.

Dominic Davis-Foster

May 15, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

```
$ python3 -m pip install dist-meta --user
```

1.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install dist-meta
```

1.3 from GitHub

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

Chapter

TWO

Using `dist_meta`

dist_meta is a library that provides for access to installed package metadata, and parsers for `METADATA` and `entry_points.txt` files. The library provides similar functionality to `importlib.metadata`, entrypoints, and `email.parser`

2.1 Overview

This example demonstrates how to obtain information about an installed distribution, such as its version number, summary and requirements.

First, create a fresh virtual environment[1] and install wheel:

Unix/macOS

```
$ python3 -m virtualenv example
$ source example/bin/activate
```

Windows

```
py -m virtualenv example
.\\example\\Scripts\\activate.ps1
```

```
(example) $ pip install wheel
```

You can get the *dist_meta.distributions.Distribution* object for `wheel` with the *dist_meta.distributions.get_distribution()* function:

Unix/macOS

```
$ python3
```

Windows

```
py
```

```
>>> from dist_meta.distributions import get_distribution
>>> wheel_dist = get_distribution("wheel")
>>> wheel_dist
<Distribution('wheel', <Version('0.36.2')>)>
>>> wheel_dist.name
'wheel'
```

(continues on next page)

[1] See the Python Packaging User Guide for more details.

(continued from previous page)

```
>>> wheel_dist.version
<Version('0.36.2')>
```

2.1.1 Metadata

The metadata for `wheel` can then be obtained as follows:

```
>>> meta = wheel_dist.get_metadata()
>>> meta
<MetadataMapping({'Metadata-Version': '2.1', 'Name': 'wheel', 'Version': '0.36.2', ...
↪})>
>>> meta["Name"]
'wheel'
>>> meta["Version"]
'0.36.2'
>>> meta["License"]
'MIT'
>>> meta["Summary"]
'A built-package format for Python'
```

This is a *dist_meta.metadata_mapping.MetadataMapping* object. See the Python Packaging User Guide for details of all supported fields.

Some fields may have only a placeholder value[2], and others may be absent:

```
>>> meta["Platform"]
'UNKNOWN'
>>> meta["Obsoletes-Dist"]
Traceback (most recent call last):
KeyError: 'Obsoletes-Dist'
```

2.1.2 Requirements

The distribution's requirements (if any) can be obtained from the `'Requires-Dist'` field:

```
>>> requirements = meta.get_all("Requires-Dist")
>>> requirements
["pytest (>=3.0.0) ; extra == 'test'", "pytest-cov ; extra == 'test'"]
```

These can be converted into `packaging.requirements.Requirement` or `shippinglabel.requirements.ComparableRequirement` objects easily:

```
>>> from packaging.requirements import Requirement
>>> list(map(Requirement, requirements))
[<Requirement('pytest>=3.0.0; extra == "test"')>, <Requirement('pytest-cov; extra ==
↪"test"')>]
```

Some distributions have no requirements:

```
>>> get_distribution("pip").get_metadata().get_all("Requires-Dist", default=[])
[]
```

[2] This is a convention followed by some build tools and not part of **PEP 566**

2.1.3 Missing Distributions

If the distribution can't be found, a *dist_meta.distributions.DistributionNotFoundError* is raised:

```
>>> get_distribution("spamspamspam")
Traceback (most recent call last):
dist_meta.distributions.DistributionNotFoundError: spamspamspam
```

2.1.4 Iterating over Distributions

All installed distributions can be iterated over using the *dist_meta.distributions.iter_distributions()* function. This can be useful to find distributions which meet a particular criteria.

For example, to find all `sphinxcontrib*` distributions:

```
>>> from dist_meta.distributions import iter_distributions
>>> for distro in iter_distributions():
...     if distro.name.startswith("sphinxcontrib"):
...         print(distro)
<Distribution('sphinxcontrib_applehelp', <Version('1.0.2')>)>
<Distribution('sphinxcontrib_htmlhelp', <Version('2.0.0')>)>
<Distribution('sphinxcontrib_jsmath', <Version('1.0.1')>)>
<Distribution('sphinxcontrib_serializinghtml', <Version('1.1.5')>)>
<Distribution('sphinxcontrib_qthelp', <Version('1.0.3')>)>
<Distribution('sphinxcontrib_devhelp', <Version('1.0.2')>)>
```

2.2 Entry Points

Entry points are a mechanism for an installed distribution to advertise components it provides for discovery and used by other code. For example:

- Distributions can specify console_scripts entry points, each referring to a function. When pip installs the distribution, it will create a command-line wrapper for each entry point.
- Applications can use entry points to load plugins; e.g. Pygments (a syntax highlighting tool) can use additional lexers and styles from separately installed packages.

Entry points are arranged into groups, such as `console_scripts` or `whey.builder`.

To obtain the entry points for a *Distribution*, call its *get_entry_points()* method:

```
>>> wheel_dist
<Distribution('wheel', <Version('0.36.2')>)>
>>> entry_points = wheel_dist.get_entry_points()
>>> entry_points.keys()
dict_keys(['console_scripts', 'distutils.commands'])
```

This returns a mapping of group names (as strings) to a mapping of entry point names to values (both strings):

```
>>> from pprint import pprint
>>> pprint(entry_points)
{'console_scripts': {'wheel': 'wheel.cli:main'},
 'distutils.commands': {'bdist_wheel': 'wheel.bdist_wheel:bdist_wheel'}}
```

dist_meta.entry_points.EntryPoint objects can be constructed as follows:

```
>>> from dist_meta.entry_points import EntryPoint
>>> for ep in EntryPoint.from_mapping(entry_points["console_scripts"], group="console_
↪scripts"):
...     ep
EntryPoint(name='wheel', value='wheel.cli:main', group='console_scripts', distro=None)
```

dist_meta.entry_points.EntryPoint objects have attributes for accessing the `name`, `module` and `attribute` of the entry point:

```
>>> ep.name
'wheel'
>>> ep.value
'wheel.cli:main'
>>> ep.module
'wheel.cli'
>>> ep.attr
'main'
>>> ep.extras
[]
```

The object referred to by the entry point can be loaded using the *load()* method:

```
>>> main = ep.load()
>>> main
<function main at 0x7f4a4bcf94c0>
```

Entry points for all distributions can be obtained using *dist_meta.entry_points.get_entry_points()* and *dist_meta.entry_points.get_all_entry_points()*. The former is used to obtain entry points in a specific group, while the latter will return all entry points grouped in a dictionary.

```
>>> from dist_meta.entry_points import get_entry_points, get_all_entry_points
>>> eps = list(get_entry_points("console_scripts"))
>>> eps[0]
EntryPoint(name='tabulate', value='tabulate:_main', group='console_scripts', distro=
↪<Distribution('tabulate', <Version('0.8.9')>)>)
>>> all_eps = get_all_entry_points()
>>> all_eps.keys()
dict_keys(['pytest11', 'console_scripts', 'sphinx.html_themes', 'distutils.commands',
↪'distutils.setup_keywords', 'babel.checkers', 'babel.extractors', 'flake8.extension
↪', 'flake8.report', 'egg_info.writers', 'setuptools.finalize_distribution_options'])
```

2.3 RECORD files

The contents of `RECORD` files, which specify the contents of a distribution, can be obtained as follows:

```
>>> wheel_dist
<Distribution('wheel', <Version('0.36.2')>)>
>>> record = wheel_dist.get_record()
>>> record[2]
RecordEntry('wheel-0.36.2.dist-info/LICENSE.txt', hash=FileHash(name='sha256', value=
↪'zKniDGrx_Pv2lAjzd3aShsvuvN7TNhAMm0o_NfvmNeQ'), size=1125, distro=<Distribution(
↪'wheel', <Version('0.36.2')>)>)
```

record is a list of *dist_meta.record.RecordEntry* objects, which is a subclasses of pathlib.PurePosixPath with additional *size*, *hash* and *distro* attributes. The content of a file can be obtained using its *read_text()* or *read_bytes()*:

```
>>> print(record[2].read_text()[:100])
"wheel" copyright (c) 2012-2014 Daniel Holth <dholth@fastmail.fm> and
contributors.

The MIT License
```

If the RECORD file is absent, *get_record()* will return None.

Chapter

THREE

dist_meta.distributions

Iterate over installed distributions.

Third-party distributions are installed into Python's `site-packages` directory with tools such as pip. Distributions must have a `*.dist-info` directory (as defined by **PEP 566**) to be discoverable.

Functions:

get_distribution(name[, path])	Returns a *Distribution* instance for the distribution with the given name.
iter_distributions([path])	Returns an iterator over installed distributions on `path`.
packages_distributions([path])	Returns a mapping of top-level packages to a list of distributions which provide them.

Classes:

DistributionType()	Abstract base class for *Distribution*-like objects.
Distribution(name, version, path)	Represents an installed Python distribution.
WheelDistribution(name, version, path, wheel_zip)	Represents a Python distribution in wheel form.

Exceptions:

DistributionNotFoundError	Raised when a distribution cannot be located.

Data:

_DT	Invariant `TypeVar` bound to *dist_meta.distributions.DistributionType*.

get_distribution(*name*, *path=None*)

Returns a *Distribution* instance for the distribution with the given name.

Parameters

- **name** (`str`)
- **path** (`Optional`[`Iterable`[`Union`[`str`, `Path`, `PathLike`]]]) – The directories entries to search for distributions in. Default `sys.path`.

Return type *Distribution*

iter_distributions(*path=None*)

Returns an iterator over installed distributions on `path`.

Parameters **path** (`Optional[Iterable[Union[str, Path, PathLike]]]`) – The directories entries to search for distributions in. Default `sys.path`.

Return type `Iterator[Distribution]`

packages_distributions(*path=None*)

Returns a mapping of top-level packages to a list of distributions which provide them.

The same top-level package may be provided by multiple distributions, especially in the case of namespace packages.

Parameters **path** (`Optional[Iterable[Union[str, Path, PathLike]]]`) – The directories entries to search for distributions in. Default `sys.path`.

New in version 0.7.0.

Example:

```
>>> import collections.abc
>>> pkgs = packages_distributions()
>>> all(isinstance(dist, collections.abc.Sequence) for dist in pkgs.values())
True
```

Return type `Mapping[str, List[str]]`

class DistributionType

Abstract base class for *Distribution*-like objects.

Changed in version 0.3.0: Previously was a `Union` representing *Distribution* and *WheelDistribution*. Now a common base class for those two classes, and custom classes providing the same API

This class implements most of the `collections.namedtuple()` API. Subclasses must implement `_fields` (as a tuple of field names) and the `tuple` interface (specifically `__iter__` and `__getitem__`).

Attributes:

name	The name of the distribution.
version	The version number of the distribution.
_fields	A tuple of field names for the "namedtuple".
_field_defaults	A mapping of field names to default values.

Methods:

read_file(filename)	Read a file from the `*.dist-info` directory and return its content.
has_file(filename)	Returns whether the `*.dist-info` directory contains a file named `filename`.
_asdict()	Return a new dict which maps field names to their values.
_replace(**kwargs)	Make a new *DistributionType* object, of the same type as this one, replacing the specified fields with new values.

continues on next page

Table 6 – continued from previous page

_make(iterable)	Make a new *DistributionType* object, of the same type as this one, from a sequence or iterable.
get_entry_points()	Returns a mapping of entry point groups to entry points.
get_metadata()	Returns the content of the `*.dist-info/METADATA` file.
get_wheel()	Returns the content of the `*.dist-info/WHEEL` file, or `None` if the file does not exist.
get_record()	Returns the parsed content of the `*.dist-info/RECORD` file, or `None` if the file does not exist.
__repr__()	Returns a string representation of the *DistributionType*.

name
Type: `str`

The name of the distribution. No normalization is performed.

version
Type: `Version`

The version number of the distribution.

_fields
Type: `Tuple[str, ...]`

A tuple of field names for the "namedtuple".

_field_defaults
Type: `Dict[str, Any]`

A mapping of field names to default values.

abstract read_file(*filename*)
Read a file from the `*.dist-info` directory and return its content.

Parameters **filename** (`str`)

Return type `str`

abstract has_file(*filename*)
Returns whether the `*.dist-info` directory contains a file named `filename`.

Parameters **filename** (`str`)

Return type `bool`

_asdict()
Return a new dict which maps field names to their values.

Return type `Dict[str, Any]`

_replace(***kwargs*)
Make a new *DistributionType* object, of the same type as this one, replacing the specified fields with new values.

Parameters **iterable**

Return type *~_DT*

classmethod _make(*iterable*)
Make a new *DistributionType* object, of the same type as this one, from a sequence or iterable.

Parameters **iterable**

Return type *~_DT*

get_entry_points()
Returns a mapping of entry point groups to entry points.

Entry points in the group are contained in a dictionary mapping entry point names to objects.

dist_meta.entry_points.EntryPoint objects can be constructed as follows:

```
for name, epstr in distro.get_entry_points().get("console_scripts", {}).
↪items():
    EntryPoint(name, epstr)
```

Return type Dict[str, Dict[str, str]]

get_metadata()
Returns the content of the `*.dist-info/METADATA` file.

Return type *MetadataMapping*

get_wheel()
Returns the content of the `*.dist-info/WHEEL` file, or None if the file does not exist.

The file will only be present if the distribution was installed from a wheel.

Return type Optional[*MetadataMapping*]

get_record()
Returns the parsed content of the `*.dist-info/RECORD` file, or None if the file does not exist.

Return type Optional[List[*RecordEntry*]]

Returns A *dist_meta.record.RecordEntry* object for each line in the record (i.e. each file in the distribution). This includes files in the `*.dist-info` directory.

__repr__()
Returns a string representation of the *DistributionType*.

Return type str

namedtuple Distribution(*name*, *version*, *path*)
Bases: *DistributionType*

Represents an installed Python distribution.

Fields

0) **name** (str) – The name of the distribution.
1) **version** (Version) – The version number of the distribution.
2) **path** (PathPlus) – The path to the `*.dist-info` directory in the file system.

classmethod from_path(*path*)
Construct a *Distribution* from a filesystem path to the `*.dist-info` directory.

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

Return type *Distribution*

read_file(*filename*)
Read a file from the `*.dist-info` directory and return its content.

Parameters **filename** (str)

Return type str

has_file(*filename*)
Returns whether the `*.dist-info` directory contains a file named `filename`.

Parameters **filename** (str)

Return type bool

get_record()
Returns the parsed content of the `*.dist-info/RECORD` file, or None if the file does not exist.

Return type Optional[List[*RecordEntry*]]

Returns A *dist_meta.record.RecordEntry* object for each line in the record (i.e. each file in the distribution). This includes files in the `*.dist-info` directory.

namedtuple WheelDistribution(*name*, *version*, *path*, *wheel_zip*)
Bases: *DistributionType*

Represents a Python distribution in wheel form.

Fields

0) **name** (str) – The name of the distribution.
1) **version** (Version) – The version number of the distribution.
2) **path** (PathPlus) – The path to the `.whl` file.
3) **wheel_zip** (ZipFile) – The opened zip file.

A *WheelDistribution* can be used as a contextmanager, which will close the underlying `zipfile.ZipFile` when exiting the `with` block.

classmethod from_path(*path*, ***kwargs*)
Construct a *WheelDistribution* from a filesystem path to the `.whl` file.

Parameters

- **path** (Union[str, Path, PathLike])
- ****kwargs** – Additional keyword arguments passed to `zipfile.ZipFile`.

Return type *WheelDistribution*

read_file(*filename*)
Read a file from the `*.dist-info` directory and return its content.

Parameters **filename** (`str`)

Return type `str`

has_file(*filename*)
Returns whether the `*.dist-info` directory contains a file named `filename`.

Parameters **filename** (`str`)

Return type `bool`

get_wheel()
Returns the content of the `*.dist-info/WHEEL` file.

Raises **FileNotFoundError** – if the file does not exist.

Return type *MetadataMapping*

get_record()
Returns the parsed content of the `*.dist-info/RECORD` file, or `None` if the file does not exist.

Return type `List`[*RecordEntry*]

Returns A *dist_meta.record.RecordEntry* object for each line in the record (i.e. each file in the distribution). This includes files in the `*.dist-info` directory.

Raises **FileNotFoundError** – if the file does not exist.

exception DistributionNotFoundError
Bases: `ValueError`

Raised when a distribution cannot be located.

_DT = TypeVar(_DT, bound=DistributionType)
Type: `TypeVar`

Invariant `TypeVar` bound to *dist_meta.distributions.DistributionType*.

Chapter

FOUR

dist_meta.entry_points

Parser and emitter for `entry_points.txt`.

Note: The functions in this module will only parse well-formed `entry_points.txt` files, and may return unexpected values if passed malformed input.

Functions:

lazy_load(filename)	Parse the entry points from the given file lazily.
lazy_loads(rawtext)	Parse the entry points from the given text lazily.
load(filename)	Parse the entry points from the given file.
loads(rawtext)	Parse the entry points from the given text.
dump(entry_points, filename)	Construct an `entry_points.txt` file for the given grouped entry points, and write it to `filename`.
dumps(entry_points)	Construct an `entry_points.txt` file for the given grouped entry points.
get_entry_points(group[, path])	Returns an iterator over `entrypoints.EntryPoint` objects in the given group.
get_all_entry_points([path])	Returns a mapping of entry point groups to entry points for all installed distributions.

Classes:

EntryPoint(name, value[, group, distro])	Represents a single entry point.

lazy_load(*filename*)

Parse the entry points from the given file lazily.

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

Return type Iterator[Tuple[str, Iterator[Tuple[str, str]]]]

Returns An iterator over `(group, entry_point)` tuples, where `entry_point` is an iterator over `(name, object)` tuples.

lazy_loads(*rawtext*)

Parse the entry points from the given text lazily.

Parameters **rawtext** (str)

Return type Iterator[Tuple[str, Iterator[Tuple[str, str]]]]

Returns An iterator over `(group, entry_point)` tuples, where `entry_point` is an iterator over `(name, object)` tuples.

load (*filename*)
Parse the entry points from the given file.

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

Return type Dict[str, Dict[str, str]]

Returns

A mapping of entry point groups to entry points.

Entry points in each group are contained in a dictionary mapping entry point names to objects.

dist_meta.entry_points.EntryPoint objects can be constructed as follows:

```
for name, epstr in distro.get_entry_points().get("console_scripts", {}).
↪items():
    EntryPoint(name, epstr)
```

loads (*rawtext*)
Parse the entry points from the given text.

Parameters **rawtext** (str)

Return type Dict[str, Dict[str, str]]

Returns

A mapping of entry point groups to entry points.

Entry points in each group are contained in a dictionary mapping entry point names to objects.

dist_meta.entry_points.EntryPoint objects can be constructed as follows:

```
for name, epstr in distro.get_entry_points().get("console_scripts", {}).
↪items():
    EntryPoint(name, epstr)
```

dump (*entry_points*, *filename*)
Construct an `entry_points.txt` file for the given grouped entry points, and write it to `filename`.

Parameters

- **entry_points** (Union[Dict[str, Dict[str, str]], Dict[str, Sequence[*EntryPoint*]]]) – A mapping of entry point groups to entry points.

  Entry points in each group are contained in a dictionary mapping entry point names to objects, or in a list of *EntryPoint* objects.

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

Return type int

dumps (*entry_points*)
Construct an `entry_points.txt` file for the given grouped entry points.

Parameters **entry_points** (Union[Dict[str, Dict[str, str]], Dict[str, Sequence[*EntryPoint*]]]) – A mapping of entry point groups to entry points.

Entry points in each group are contained in a dictionary mapping entry point names to objects, or in a list of *EntryPoint* objects.

Return type str

get_entry_points(*group*, *path=None*)
Returns an iterator over `entrypoints.EntryPoint` objects in the given group.

Parameters

- **group** (`str`)
- **path** (`Optional[Iterable[Union[str, Path, PathLike]]]`) – The directories entries to search for distributions in. Default `sys.path`.

Return type `Iterator[EntryPoint]`

get_all_entry_points(*path=None*)
Returns a mapping of entry point groups to entry points for all installed distributions.

Parameters **path** (`Optional[Iterable[Union[str, Path, PathLike]]]`) – The directories entries to search for distributions in. Default `sys.path`.

Return type `Dict[str, List[EntryPoint]]`

namedtuple EntryPoint(*name*, *value*, *group=None*, *distro=None*)
Bases: `NamedTuple`

Represents a single entry point.

Fields

0) **name** (`str`) – The name of the entry point.
1) **value** (`str`) – The value of the entry point, in the form `module.submodule:attribute`.
2) **group** (`Optional[str]`) – The group the entry point belongs to.
3) **distro** (`Optional[Distribution]`) – The distribution the entry point belongs to.

load()
Load the object referred to by this entry point.

If only a module is indicated by the value, return that module. Otherwise, return the named object.

Return type `object`

property extras
Returns the list of extras associated with the entry point.

Return type `List[str]`

property module
The module component of *self.value*.

Return type `str`

property attr
The object/attribute component of *self.value*.

Return type `str`

classmethod from_mapping(*mapping*, ***, *group=None*, *distro=None*)
Returns a list of *EntryPoint* objects constructed from values in `mapping`.

Parameters

- **mapping** (`Mapping`[`str`, `str`]) – A mapping of entry point names to values, where values are in the form `module.submodule:attribute`.
- **group** (`Optional`[`str`]) – The group the entry points belong to. Default `None`.
- **distro** (`Optional`[*Distribution*]) – The distribution the entry points belong to. Default `None`.

Return type `List`[*EntryPoint*]

__repr__()
Return a nicely formatted representation string

Chapter

FIVE

dist_meta.metadata

Parse and create `*dist-info/METADATA` files.

Exceptions:

MissingFieldError	Raised when a required field is missing.

Functions:

dump(fields, filename)	Construct Python core metadata from the given fields, and write it to `filename`.
dumps(fields)	Construct Python core metadata from the given fields.
load(filename)	Parse Python core metadata from the given file.
loads(rawtext)	Parse Python core metadata from the given string.

exception MissingFieldError

Bases: `ValueError`

Raised when a required field is missing.

dump (*fields*, *filename*)

Construct Python core metadata from the given fields, and write it to `filename`.

Parameters

- **fields** (*MetadataMapping*)
- **filename** (`Union`[`str`, `Path`, `PathLike`])

Return type `int`

dumps (*fields*)

Construct Python core metadata from the given fields.

Parameters **fields** (*MetadataMapping*)

Return type `str`

Changed in version 0.4.0: Added support for the License-Expression and License-File options proposed by **PEP 639**.

load(*filename*)
Parse Python core metadata from the given file.

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

Return type *MetadataMapping*

Returns A mapping of the metadata fields, and the long description

loads(*rawtext*)
Parse Python core metadata from the given string.

Parameters **rawtext** (str)

Return type *MetadataMapping*

Returns A mapping of the metadata fields, and the long description

Chapter

SIX

dist_meta.metadata_mapping

collections.abc.MutableMapping which supports duplicate, case-insensitive keys.

Caution: These are pretty low-level classes. You probably don't need to use these directly unless you're customising the METADATA file creation or parsing.

Classes:

MetadataMapping()	Provides a MutableMapping interface to a list of fields, such as those used for Python core metadata.
MetadataEmitter(fields)	Used to construct METADATA, WHEEL and other email field-like files.

class MetadataMapping

Bases: MutableMapping[str, str]

Provides a MutableMapping interface to a list of fields, such as those used for Python core metadata.

See also: email.message.Message and email.message.EmailMessage

Implements the MutableMapping interface, which assumes there is exactly one occurrence of the field per mapping. Some fields do in fact appear multiple times, and for those fields you must use the *get_all()* method to obtain all values for that field.

Methods:

__len__()	Return the total number of keys, including duplicates.
__getitem__(name)	Get a field value.
__setitem__(name, val)	Set the value of a field.
__delitem__(name)	Delete all occurrences of a field, if present.
__contains__(name)	Returns whether name is in the *MetadataMapping*.
__iter__()	Returns an iterator over the keys in the *MetadataMapping*.
keys()	Return a list of all field *names*.
values()	Return a list of all field *values*.
items()	Get all the fields and their values.
get(name[, default])	Get a field value.
get_all(name[, default])	Return a list of all the values for the named field.
__repr__()	Return a string representation of the *MetadataMapping*.
replace(name, value)	Replace the value of the first matching field, retaining header order and case.

__len__()
Return the total number of keys, including duplicates.

Return type int

__getitem__(*name*)
Get a field value.

Note: If the field appears multiple times, exactly which occurrence gets returned is undefined. Use the *get_all()* method to get all values matching a field name.

Parameters **name** (str)

Return type str

__setitem__(*name*, *val*)
Set the value of a field.

Parameters

- **name** (str)
- **val** (str)

__delitem__(*name*)
Delete all occurrences of a field, if present.

Does not raise an exception if the field is missing.

Parameters **name** (str)

__contains__(*name*)
Returns whether name is in the *MetadataMapping*.

Parameters **name** (object)

Return type bool

__iter__()
Returns an iterator over the keys in the *MetadataMapping*.

Return type Iterator[str]

keys()
Return a list of all field *names*.

These will be sorted by insertion order, and may contain duplicates. Any fields deleted and re-inserted are always appended to the field list.

Return type List[str]

values()
Return a list of all field *values*.

These will be sorted by insertion order, and may contain duplicates. Any fields deleted and re-inserted are always appended to the field list.

Return type List[str]

items()
Get all the fields and their values.

These will be sorted by insertion order, and may contain duplicates. Any fields deleted and re-inserted are always appended to the field list.

Return type List[Tuple[str, str]]

get(*name*, *default=None*)
Get a field value.

Like *__getitem__()*, but returns default instead of None when the field is missing.

Note: If the field appears multiple times, exactly which occurrence gets returned is undefined. Use the *get_all()* method to get all values matching a field name.

Parameters

- **name** (str)
- **default** – Default None.

Return type str

Overloads

- *get*(**name**: str) -> Optional[str]
- *get*(**name**: str, **default**: Union[str, ~_T]) -> Union[str, ~_T]

get_all(*name*, *default=None*)
Return a list of all the values for the named field.

These will be sorted in the order they appeared in the original message, and may contain duplicates. Any fields deleted and re-inserted are always appended to the field list.

If no such fields exist, default is returned.

Parameters

- **name** (str)
- **default** – Default None.

Overloads

- *get_all*(**name**: str) -> Optional[List[str]]
- *get_all*(**name**: str, **default**: Union[str, ~_T]) -> Union[List[str], ~_T]

__repr__()
Return a string representation of the *MetadataMapping*.

Return type str

replace(*name*, *value*)
Replace the value of the first matching field, retaining header order and case.

Raises **KeyError** – If no matching field was found.

class MetadataEmitter(*fields*)

Bases: StringList

Used to construct METADATA, WHEEL and other email field-like files.

Parameters **fields** (*MetadataMapping*) – The fields the file is being constructed from.

Methods:

add_single(field_name)	Add a single value for the field with the given name.
add_multiple(field_name)	Add all values for the "multiple use" field with the given name.
add_body(body)	Add a body to the file.

add_single(*field_name*)

Add a single value for the field with the given name.

Parameters **field_name** (str)

add_multiple(*field_name*)

Add all values for the "multiple use" field with the given name.

Parameters **field_name** (str)

add_body(*body*)

Add a body to the file.

In an email message this is the message content itself.

Parameters **body** (str)

Chapter

SEVEN

dist_meta.record

Classes to model parts of RECORD files.

Classes:

FileHash(name, value)	Represents a checksum for a file in a RECORD file, or as the URL fragment in a **PEP 503** repository URL.
RecordEntry(path[, hash, size, distro])	Represents a path in a distribution.

class RecordEntry(*path*, *hash=None*, *size=None*, *distro=None*)

Bases: PurePosixPath

Represents a path in a distribution.

Parameters

- **path** (Union[str, Path, PathLike]) – The path to the file in the distribution, relative to the distribution root (i.e. the site-packages directory).
- **hash** (Optional[*FileHash*]) – The hash/checksum of the file. Default None.
- **size** (Optional[int]) – The size of the file. Default None.
- **distro** (Optional[*Distribution*]) – The distribution the file belongs to. Default None.

Note: Path operations (joinpath(), parent etc.) will return a standard pathlib.PurePosixPath object without the extended attributes of this class.

Methods:

__repr__()	Return a string representation of the *RecordEntry*.
as_record_entry()	Returns an entry for a RECORD file, in the form <name>,<hash>,<size>.
from_record_entry(entry[, distro])	Construct a *RecordEntry* from a line in a RECORD file, in the form <name>,<hash>,<size>.
read_bytes()	Open the file in bytes mode, read it, and close the file.
read_text([encoding, errors])	Open the file in text mode, read it, and close the file.

Attributes:

distro	The distribution the file belongs to.
hash	The hash/checksum of the file.
size	The size of the file.

__repr__()
Return a string representation of the *RecordEntry*.

Return type str

as_record_entry()
Returns an entry for a RECORD file, in the form <name>,<hash>,<size>.

Return type str

distro
Type: Optional[*Distribution*]

The distribution the file belongs to.

classmethod from_record_entry(*entry*, *distro=None*)
Construct a *RecordEntry* from a line in a RECORD file, in the form <name>,<hash>,<size>.

New in version 0.2.0.

Parameters

- **entry** (str)
- **distro** (Optional[*Distribution*]) – The distribution the RECORD file belongs to. Optional. Default None.

Return type *RecordEntry*

hash
Type: Optional[*FileHash*]

The hash/checksum of the file.

read_bytes()
Open the file in bytes mode, read it, and close the file.

Return type bytes

Returns The content of the file.

> **Attention:** This operation requires a value for *self.distro*.

read_text(*encoding='UTF-8'*, *errors=None*)
Open the file in text mode, read it, and close the file.

Parameters

- **encoding** (Optional[str]) – The encoding to write to the file in. Default 'UTF-8'.
- **errors** (Optional[str]) – Default None.

Return type str

Returns The content of the file.

> **Attention:** This operation requires a value for *self.distro*.

size
Type: Optional[int]

The size of the file.

namedtuple FileHash(*name*, *value*)
Bases: NamedTuple

Represents a checksum for a file in a RECORD file, or as the URL fragment in a **PEP 503** repository URL.

Fields

0) **name** (str) – The name of the hash algorithm.
1) **value** (str) – The urlsafe_b64encode() 'd hexdigest of the hash.

digest()
Returns the digest of the hash.

This is a bytes object which may contain bytes in the whole range from 0 to 255.

Return type bytes

classmethod from_hash(*the_hash*)
Construct a *FileHash* object from a hashlib hash object.

Parameters **the_hash** (hashlib.HASH)

Return type *FileHash*

classmethod from_string(*string*)
Constructs a *FileHash* from a string in the form <name>=<value>.

Parameters **string** (str)

Return type *FileHash*

hexdigest()
Like *self.digest()* except the digest is returned as a string object of double length, containing only hexadecimal digits.

This may be used to exchange the value safely in email or other non-binary environments.

Return type str

to_string()
Returns the *FileHash* as a string, in the form <name>=<value>.

Return type str

Chapter

EIGHT

dist_meta.wheel

Parse and create `*dist-info/WHEEL` files.

Functions:

dump(fields, filename)	Construct a `WHEEL` file from the given fields, and write it to `filename`.
dumps(fields)	Construct a `WHEEL` file from the given fields.
load(filename)	Parse a `WHEEL` file from the given file.
loads(rawtext)	Parse a `WHEEL` file from the given string.
parse_generator_string(generator)	Parse a generator string into its name and version.

dump (*fields*, *filename*)

Construct a `WHEEL` file from the given fields, and write it to `filename`.

Parameters

- **fields** (`Union`[`Mapping`[`str`, `Any`], *MetadataMapping*]) – May be a conventional mapping, with `Root-Is-Purelib` as a boolean and `Tag` as a list of strings.
- **filename** (`Union`[`str`, `Path`, `PathLike`])

Return type `int`

dumps (*fields*)

Construct a `WHEEL` file from the given fields.

Parameters **fields** (`Union`[`Mapping`[`str`, `Any`], *MetadataMapping*]) – May be a conventional mapping, with `Root-Is-Purelib` as a boolean and `Tag` as a list of strings.

Return type `str`

load (*filename*)

Parse a `WHEEL` file from the given file.

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

Return type *MetadataMapping*

Returns A mapping of the metadata fields, and the long description

loads (*rawtext*)

Parse a `WHEEL` file from the given string.

Parameters **rawtext** (`str`)

Return type *MetadataMapping*

Returns A mapping of the metadata fields, and the long description

parse_generator_string(*generator*)

Parse a generator string into its name and version.

Common forms include:

- `name (version)`
- `name version`
- `name`

New in version 0.6.0.

Parameters **generator** (`str`) – The raw generator string (the `Generator` field in `WHEEL`).

Return type `Tuple[str, Optional[str]]`

Returns A tuple of the generator name and its version. The version may be `None` if no version could be found.

Python Module Index

d

Index

Symbols

A

D

E

F

G

H

I

K

L

M

N

P

R

S

T

V

W