

---

# **dist-meta**

***Release 0.8.0***

**Parse and create Python distribution metadata.**

**Dominic Davis-Foster**

**Apr 25, 2023**



# Contents

<b>1</b>	<b>Installation</b>	<b>1</b>
1.1	from PyPI . . . . .	1
1.2	from Anaconda . . . . .	1
1.3	from GitHub . . . . .	1
<b>2</b>	<b>Using <code>dist_meta</code></b>	<b>3</b>
2.1	Overview . . . . .	3
2.2	Entry Points . . . . .	5
2.3	RECORD files . . . . .	6
<b>3</b>	<b><code>dist_meta.distributions</code></b>	<b>9</b>
3.1	<code>get_distribution</code> . . . . .	9
3.2	<code>iter_distributions</code> . . . . .	10
3.3	<code>packages_distributions</code> . . . . .	10
3.4	<code>DistributionType</code> . . . . .	10
3.5	<code>Distribution</code> . . . . .	12
3.6	<code>WheelDistribution</code> . . . . .	13
3.7	<code>DistributionNotFoundError</code> . . . . .	14
3.8	<code>_DT</code> . . . . .	14
<b>4</b>	<b><code>dist_meta.entry_points</code></b>	<b>15</b>
4.1	<code>lazy_load</code> . . . . .	15
4.2	<code>lazy_loads</code> . . . . .	15
4.3	<code>load</code> . . . . .	16
4.4	<code>loads</code> . . . . .	16
4.5	<code>dump</code> . . . . .	16
4.6	<code>dumps</code> . . . . .	16
4.7	<code>get_entry_points</code> . . . . .	17
4.8	<code>get_all_entry_points</code> . . . . .	17
4.9	<code>EntryPoint</code> . . . . .	17
<b>5</b>	<b><code>dist_meta.metadata</code></b>	<b>19</b>
5.1	<code>MissingFieldError</code> . . . . .	19
5.2	<code>dump</code> . . . . .	19
5.3	<code>dumps</code> . . . . .	19
5.4	<code>load</code> . . . . .	20
5.5	<code>loads</code> . . . . .	20
<b>6</b>	<b><code>dist_meta.metadata_mapping</code></b>	<b>21</b>
6.1	<code>MetadataMapping</code> . . . . .	21
6.2	<code>MetadataEmitter</code> . . . . .	24

<b>7</b>	<b><code>dist_meta.record</code></b>	<b>25</b>
7.1	<code>RecordEntry</code> . . . . .	25
7.2	<code>FileHash</code> . . . . .	27
<b>8</b>	<b><code>dist_meta.wheel</code></b>	<b>29</b>
8.1	<code>dump</code> . . . . .	29
8.2	<code>dumps</code> . . . . .	29
8.3	<code>load</code> . . . . .	29
8.4	<code>loads</code> . . . . .	29
8.5	<code>parse_generator_string</code> . . . . .	30
	<b>Python Module Index</b>	<b>31</b>
	<b>Index</b>	<b>33</b>

## 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
```



## 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<sup>1</sup> 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)

---

<sup>1</sup> 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<sup>2</sup>, 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=[])
[]
```

---

<sup>2</sup> 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 `wheel.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`.



## `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:

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

### Classes:

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

### Exceptions:

<code>DistributionNotFoundError</code>	Raised when a distribution cannot be located.
--	---

### Data:

<code>_DT</code>	Invariant <code>TypeVar</code> bound to <code>dist_meta.distributions.DistributionType</code> .
------------------	---

**`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:**

<code>name</code>	The name of the distribution.
<code>version</code>	The version number of the distribution.
<code>_fields</code>	A tuple of field names for the “namedtuple”.
<code>_field_defaults</code>	A mapping of field names to default values.

**Methods:**

<code>read_file(filename)</code>	Read a file from the <code>*.dist-info</code> directory and return its content.
<code>has_file(filename)</code>	Returns whether the <code>*.dist-info</code> directory contains a file named <code>filename</code> .
<code>_asdict()</code>	Return a new dict which maps field names to their values.
<code>_replace(**kwargs)</code>	Make a new <i>DistributionType</i> 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

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

**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`.

## `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:

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

### Classes:

<code>EntryPoint(name, value[, group, distro])</code>	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

## `dist_meta.metadata`

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

### Exceptions:

<code>MissingFieldError</code>	Raised when a required field is missing.
--------------------------------	--

### Functions:

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



## `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:

<code>MetadataMapping()</code>	Provides a <code>MutableMapping</code> interface to a list of fields, such as those used for Python <code>core metadata</code> .
<code>MetadataEmitter(fields)</code>	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:

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

---

<code>add_single</code> ( <i>field_name</i> )	Add a single value for the field with the given name.
<code>add_multiple</code> ( <i>field_name</i> )	Add all values for the “multiple use” field with the given name.
<code>add_body</code> ( <i>body</i> )	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`)

## `dist_meta.record`

Classes to model parts of RECORD files.

### Classes:

<code>FileHash(name, value)</code>	Represents a checksum for a file in a RECORD file, or as the URL fragment in a <b>PEP 503</b> repository URL.
<code>RecordEntry(path[, hash, size, distro])</code>	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:

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

### Attributes:

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

<b>Attention:</b> This operation requires a value for <i>self.distro</i> .
--

**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`





## `dist_meta.wheel`

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

### Functions:

<code>dump(fields, filename)</code>	Construct a WHEEL file from the given fields, and write it to <code>filename</code> .
<code>dumps(fields)</code>	Construct a WHEEL file from the given fields.
<code>load(filename)</code>	Parse a WHEEL file from the given file.
<code>loads(rawtext)</code>	Parse a WHEEL file from the given string.
<code>parse_generator_string(generator_string)</code>	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

- `dist_meta`, [3](#)
- `dist_meta.distributions`, [9](#)
- `dist_meta.entry_points`, [15](#)
- `dist_meta.metadata`, [19](#)
- `dist_meta.metadata_mapping`, [21](#)
- `dist_meta.record`, [25](#)
- `dist_meta.wheel`, [29](#)



## Symbols

\_DT (in module *dist\_meta.distributions*), 14  
 \_\_contains\_\_() (MetadataMapping method), 22  
 \_\_delitem\_\_() (MetadataMapping method), 22  
 \_\_getitem\_\_() (MetadataMapping method), 22  
 \_\_iter\_\_() (MetadataMapping method), 22  
 \_\_len\_\_() (MetadataMapping method), 21  
 \_\_repr\_\_() (DistributionType method), 12  
 \_\_repr\_\_() (EntryPoint method), 18  
 \_\_repr\_\_() (MetadataMapping method), 23  
 \_\_repr\_\_() (RecordEntry method), 25  
 \_\_setitem\_\_() (MetadataMapping method), 22  
 \_asdict() (DistributionType method), 11  
 \_field\_defaults (DistributionType attribute), 11  
 \_fields (DistributionType attribute), 11  
 \_make() (DistributionType class method), 12  
 \_replace() (DistributionType method), 11

## A

add\_body() (MetadataEmitter method), 24  
 add\_multiple() (MetadataEmitter method), 24  
 add\_single() (MetadataEmitter method), 24  
 as\_record\_entry() (RecordEntry method), 26  
 attr() (EntryPoint property), 17

## D

digest() (FileHash method), 27  
 dist\_meta  
     module, 3  
 dist\_meta.distributions  
     module, 9  
 dist\_meta.entry\_points  
     module, 15  
 dist\_meta.metadata  
     module, 19  
 dist\_meta.metadata\_mapping  
     module, 21  
 dist\_meta.record  
     module, 25  
 dist\_meta.wheel  
     module, 29  
 Distribution (namedtuple in  
     *dist\_meta.distributions*), 12

    name (namedtuple field), 12  
     path (namedtuple field), 12  
     version (namedtuple field), 12  
 DistributionNotFoundError, 14  
 DistributionType (class in  
     *dist\_meta.distributions*), 10  
 distro (namedtuple field)  
     EntryPoint (namedtuple in  
         *dist\_meta.entry\_points*), 17  
 distro (RecordEntry attribute), 26  
 dump() (in module *dist\_meta.entry\_points*), 16  
 dump() (in module *dist\_meta.metadata*), 19  
 dump() (in module *dist\_meta.wheel*), 29  
 dumps() (in module *dist\_meta.entry\_points*), 16  
 dumps() (in module *dist\_meta.metadata*), 19  
 dumps() (in module *dist\_meta.wheel*), 29

## E

EntryPoint (namedtuple in *dist\_meta.entry\_points*),  
     17  
     distro (namedtuple field), 17  
     group (namedtuple field), 17  
     name (namedtuple field), 17  
     value (namedtuple field), 17  
 extras() (EntryPoint property), 17

## F

FileHash (namedtuple in *dist\_meta.record*), 27  
     name (namedtuple field), 27  
     value (namedtuple field), 27  
 from\_hash() (FileHash class method), 27  
 from\_mapping() (EntryPoint class method), 18  
 from\_path() (Distribution class method), 12  
 from\_path() (WheelDistribution class method), 13  
 from\_record\_entry() (RecordEntry class  
     method), 26  
 from\_string() (FileHash class method), 27

## G

get() (MetadataMapping method), 23  
 get\_all() (MetadataMapping method), 23  
 get\_all\_entry\_points() (in module  
     *dist\_meta.entry\_points*), 17

`get_distribution()` (in module *dist\_meta.distributions*), 9  
`get_entry_points()` (*DistributionType* method), 12  
`get_entry_points()` (in module *dist\_meta.entry\_points*), 17  
`get_metadata()` (*DistributionType* method), 12  
`get_record()` (*Distribution* method), 13  
`get_record()` (*DistributionType* method), 12  
`get_record()` (*WheelDistribution* method), 14  
`get_wheel()` (*DistributionType* method), 12  
`get_wheel()` (*WheelDistribution* method), 14  
`group` (*namedtuple* field)  
    *EntryPoint* (*namedtuple* in *dist\_meta.entry\_points*), 17

## H

`has_file()` (*Distribution* method), 13  
`has_file()` (*DistributionType* method), 11  
`has_file()` (*WheelDistribution* method), 14  
`hash` (*RecordEntry* attribute), 26  
`hexdigest()` (*FileHash* method), 27

## I

`items()` (*MetadataMapping* method), 23  
`iter_distributions()` (in module *dist\_meta.distributions*), 9

## K

`keys()` (*MetadataMapping* method), 22

## L

`lazy_load()` (in module *dist\_meta.entry\_points*), 15  
`lazy_loads()` (in module *dist\_meta.entry\_points*), 15  
`load()` (*EntryPoint* method), 17  
`load()` (in module *dist\_meta.entry\_points*), 15  
`load()` (in module *dist\_meta.metadata*), 20  
`load()` (in module *dist\_meta.wheel*), 29  
`loads()` (in module *dist\_meta.entry\_points*), 16  
`loads()` (in module *dist\_meta.metadata*), 20  
`loads()` (in module *dist\_meta.wheel*), 29

## M

*MetadataEmitter* (class in *dist\_meta.metadata\_mapping*), 23  
*MetadataMapping* (class in *dist\_meta.metadata\_mapping*), 21  
*MissingFieldError*, 19  
module  
    *dist\_meta*, 3  
    *dist\_meta.distributions*, 9  
    *dist\_meta.entry\_points*, 15  
    *dist\_meta.metadata*, 19

*dist\_meta.metadata\_mapping*, 21  
    *dist\_meta.record*, 25  
    *dist\_meta.wheel*, 29  
`module()` (*EntryPoint* property), 17

## N

`name` (*DistributionType* attribute), 11  
`name` (*namedtuple* field)  
    *Distribution* (*namedtuple* in *dist\_meta.distributions*), 12  
    *EntryPoint* (*namedtuple* in *dist\_meta.entry\_points*), 17  
    *FileHash* (*namedtuple* in *dist\_meta.record*), 27  
    *WheelDistribution* (*namedtuple* in *dist\_meta.distributions*), 13

## P

`packages_distributions()` (in module *dist\_meta.distributions*), 10  
`parse_generator_string()` (in module *dist\_meta.wheel*), 29  
`path` (*namedtuple* field)  
    *Distribution* (*namedtuple* in *dist\_meta.distributions*), 12  
    *WheelDistribution* (*namedtuple* in *dist\_meta.distributions*), 13  
*Python Enhancement Proposals*  
    PEP 427, 9, 12, 13  
    PEP 503, 25, 27  
    PEP 566, 4, 9  
    PEP 639, 19

## R

`read_bytes()` (*RecordEntry* method), 26  
`read_file()` (*Distribution* method), 13  
`read_file()` (*DistributionType* method), 11  
`read_file()` (*WheelDistribution* method), 13  
`read_text()` (*RecordEntry* method), 26  
*RecordEntry* (class in *dist\_meta.record*), 25  
`replace()` (*MetadataMapping* method), 23

## S

`size` (*RecordEntry* attribute), 27

## T

`to_string()` (*FileHash* method), 27

## V

`value` (*namedtuple* field)  
    *EntryPoint* (*namedtuple* in *dist\_meta.entry\_points*), 17  
    *FileHash* (*namedtuple* in *dist\_meta.record*), 27  
`values()` (*MetadataMapping* method), 22

version (*DistributionType* attribute), [11](#)  
version (*namedtuple* field)  
    Distribution (*namedtuple* in  
        *dist\_meta.distributions*), [12](#)  
    WheelDistribution (*namedtuple* in  
        *dist\_meta.distributions*), [13](#)

## W

wheel\_zip (*namedtuple* field)  
    WheelDistribution (*namedtuple* in  
        *dist\_meta.distributions*), [13](#)  
WheelDistribution (*namedtuple* in  
    *dist\_meta.distributions*), [13](#)  
    name (*namedtuple* field), [13](#)  
    path (*namedtuple* field), [13](#)  
    version (*namedtuple* field), [13](#)  
    wheel\_zip (*namedtuple* field), [13](#)