Documentation sitecustomize-entrypoints

Release 1.0.0" # poetry-dynamic-versioning substitutes this

Darsstar

Apr 10, 2023

Contents

Testing Linting Read the Docs PyPi Package MIT License

1 sitecustomize-entrypoints

1.1 Overview

`sitecustomize-entrypoints` is a library that installs a python-module called `sitecustomize`, and allows you to define and register any callable as a `sitecustomize`-entrypoint in your project's `setup.py` or `pyproject.toml`.

These callables will be then executed automatically whenever `sitecustomize` is imported during python-startup. This provides a simple & modular way to patch or extend the functionality of any python-code installed in your environment without modifying its source code.

`sitecustomize` is a special python-module that can be used to customize the python environment at startup. When the python-interpreter starts, it looks for a `sitecustomize.py`-file in the site-packages directory or any other directory specified in the `PYTHONPATH`. The `sitecustomize.py`-file is executed before any other Python code, allowing you to make customizations that will be applied to the entire python environment. For more information, check the official site.py-documentation.

1.2 Installation

You can install `sitecustomize-entrypoints` using pip:

```
> bin/pip install sitecustomize-entrypoints
```

Or add it to your poetry-based project:

```
> poetry add sitecustomize-entrypoints
```

1.3 Usage

To use `sitecustomize-entrypoints`, you need to define and register one or more entrypoints in your project's `setup.py` or `pyproject.toml` file. Here's an example:

```
[tool.poetry.plugins."sitecustomize"]
"my-action" = "my_project.action:my_action"
```

In this example, we're registering an entrypoint called `my-action` in the `sitecustomize`-group. This entrypoint points to the `my_action`-function in the `my_project.actions`-module.

Once you've registered your entrypoints, they will be executed **automatically** when the `sitecustomize`-module is imported.

```

import sitecustomize
```

1.4 Ordering

Entrypoints are sorted by name.

The ordering in which the entrypoints are defined in your `setup.py` or `pyproject.toml` are unfortunately not the order in which they are registered internally.

The entrypoints are first ordered alphanumerically by name.

So defined entrypoints are re-ordered from

```
[tool.poetry.plugins."sitecustomize"]
foo = "my_project.action:action_foo"
bar = "my_project.action:action_bar"
```

into

```
[tool.poetry.plugins."sitecustomize"]
bar = "my_project.action:action_bar"
foo = "my_project.action:action_foo"
```

This can cause issues when there are dependencies between your entrypoints.

TIP: You can use integer-prefixes to the name to enforce the ordering in which you define your entrypoints.:

```
[tool.poetry.plugins."sitecustomize"]
10-foo = "my_project.action:action_foo"
20-bar = "my_project.action:action_bar"
```

1.5 Cancel entrypoints registered by third-party packages

Imagine a third-party package has registered an entrypoint as follows:

```
[tool.poetry.plugins."sitecustomize"]
their-action = "external_module:their_action"
```

But it does not match the ordering you'd like.

You can cancel the execution of this plugin, by registering an entrypoint with the same name in your own `setup.py` or `pyproject.toml` file, and pointing to the no-action `sitecustomize.cancel`-function.

```
[tool.poetry.plugins."sitecustomize"]
 their-action = "sitecustomize:cancel"
```

By registering an entrypoint with the **same name** it overrides the previous registered action. Subsequently you can re-register this entrypoint by a different name to change the order of execution.

1.6 Overriding and re-ordering entrypoints

You can override and re-order registered entrypoints by defining your own entrypoints with the same name in your setup.py or pyproject.toml file. When `sitecustomize` executes registered entrypoints, it uses a FIFO-approach (first in, first out): Only the last registered entrypoint with a given name is executed. Any prior entrypoints with the same name are filtered out (overriden by the last).

To re-order entrypoints, you can use the `cancel`-function provided by `sitecustomize-entrypoints`. The `cancel`-function allows you to cancel out a previously registered entrypoint, so you can re-register it in a different order.

Here's an example to enforce a specific order of execution:

```
[tool.poetry.plugins."sitecustomize"]
"their-action = external_module:their_action"
```

```
[tool.poetry.plugins."sitecustomize"]
"their-action = sitecustomize:cancel"
"10-my-action = my_project.action:my_action"
"20-their-action = external_module:their_action"
```

In this example, we're re-ordering an entrypoint registered by a third-party module to be executed after our own. First we re-register it with the same name to override and cancel its action, and subsequently we re-register it under a different name to ensure it get executed after our own entrypoints, even if all entrypoints are sorted alphabetically

1.7 Display all registered entrypoint in order of execution

The `sitecustomize.print_entrypoints`-function provided by `sitecustomize-entrypoints` allows you to display a list of all registered entrypoints. By default, it will display all registered entrypoints in the `sitecustomize`-group in order of execution. You can optionally pass a different entrypoint group name as an argument. If you pass `filtered=True` as an argument, the function will also filter out any duplicate entrypoints and display only the last instance of each entrypoint. Here's an example usage:

```
import sitecustomize

# Display all registered entrypoints in the sitecustomize group
sitecustomize.print_entrypoints()

# More explicit equivalent
sitecustomize.print_entrypoints(group_name="sitecustomize")

# Display only the first instance of each registered entrypoint in the sitecustomize␣
↪group
sitecustomize.print_entrypoints(filtered=True)
```

2 Changelog

All notable changes to this project will be documented in this file.

2.1 Unreleased (YYYY-MM-DD)

- Add colorful badges to readme. [WouterVH]
- Add `ruff` as dev-dependency. [WouterVH]
- Add github-action for testing. [WouterVH]
- Add github-action for linting. [WouterVH]
- Use a constant `ENTRYPOINT_GROUPNAME`. [WouterVH]
- Add `print_entrypoints`-function to easily output a list an registered entrypoints. [WouterVH]
- Add `cancel`-function that you can use in your own `pyproject.toml` to override pre-existing entrypoints so you can re-order them. [WouterVH]
- Avoid executing a registered entrypoint more than once, even when it was registered multiple times. [WouterVH]

- Allow to set the order in your own `pyproject.toml`, in which any registered entrypoint executed. [WouterVH]
- Display warnings in a simplified format. [WouterVH]
- Add changelog. [WouterVH]
- Add license. [WouterVH]
- Complete project-metadata in `pyproject.toml`. [WouterVH]
- Avoid breaking when a registered entrypoint throws an `AttributeError`. [WouterVH]

2.2 1.0.0 (2023-03-10)

- Avoid breaking when a registered entrypoint throws an error when executed [WouterVH]
- Avoid breaking when a registered entrypoint cannot be found, e.g. renamed function. [WouterVH]

2.3 0.1.0 (2022-04-19)

- Package created by [Dos Moonen d.moonen@nki.nl]

3 Security Policy

3.1 Supported Versions

Use this section to tell people about which versions of your project are currently being supported with security updates.

| Version | Supported | | ——- | ————— | | 0.x | :white_check_mark: | | 1.0.x | :white_check_mark: |

3.2 Reporting a Vulnerability

This project follows a 90 day disclosure timeline.

To report a security issue, please an email security@libranet.eu with

- a description of the issue
- the steps you took to create the issue,
- affected versions
- and if known, mitigations for the issue

Our team will acknowledge receiving your email within 3 working days.

4 MIT License

5 Contributors

Special thanks for all the people who have helped on this project so far.

Append your name if you have contributed to this package. We use anti-chronological ordering (oldest on top).

- Dos Moonen <d.moonen@nki.nl>
- Wouter Vanden Hove <wouter@libranet.eu>

6 How to contribute

When contributing to this repository, please first discuss the change you wish to make via issue, email, or any other method with the owners of this repository before making a change.

Please note we have a *code of conduct*, please follow it in all your interactions with the project.

6.1 Development environment setup

Proceed to describe how to setup local development environment. e.g:

To set up a development environment, please follow these steps:

1. Clone the repo

```
git clone https://github.com/Darsstar/sitecustomize-entrypoints
```

2. Run poetry install

```
poetry install
```

6.2 Issues and feature requests

You've found a bug in the source code, a mistake in the documentation or maybe you'd like a new feature? Take a look at GitHub Discussions to see if it's already being discussed.

You can help us by submitting an issue on GitHub. Before you create an issue, make sure to search the issue archive – your issue may have already been addressed.

Please try to create bug reports that are:

- *Reproducible.* Include steps to reproduce the problem.
- *Specific.* Include as much detail as possible: which version, what environment, etc.
- *Unique.* Do not duplicate existing opened issues.
- *Scoped to a Single Bug.* One bug per report.

Even better: Submit a pull request with a fix or new feature!

How to submit a Pull Request

1. Search our repository for open or closed Pull Requests that relate to your submission. You don't want to duplicate effort.
2. Fork the project
3. Create your feature branch (`git checkout -b feat/amazing_feature`)
4. Commit your changes (`git commit -m 'feat: add amazing_feature'`) sitecustomize-entrypoints uses conventional commits, so please follow the specification in your commit messages.
5. Push to the branch (`git push origin feat/amazing_feature`)
6. Open a Pull Request

7 Contributor Covenant Code of Conduct

7.1 Our Pledge

We as members, contributors, and leaders pledge to make participation in our community a harassment-free experience for everyone, regardless of age, body size, visible or invisible disability, ethnicity, sex characteristics, gender identity and expression, level of experience, education, socio-economic status, nationality, personal appearance, race, religion, or sexual identity and orientation.

We pledge to act and interact in ways that contribute to an open, welcoming, diverse, inclusive, and healthy community.

7.2 Our Standards

Examples of behavior that contributes to a positive environment for our community include:

- Demonstrating empathy and kindness toward other people
- Being respectful of differing opinions, viewpoints, and experiences
- Giving and gracefully accepting constructive feedback
- Accepting responsibility and apologizing to those affected by our mistakes, and learning from the experience
- Focusing on what is best not just for us as individuals, but for the overall community

Examples of unacceptable behavior include:

- The use of sexualized language or imagery, and sexual attention or advances of any kind
- Trolling, insulting or derogatory comments, and personal or political attacks
- Public or private harassment
- Publishing others' private information, such as a physical or email address, without their explicit permission
- Other conduct which could reasonably be considered inappropriate in a professional setting

7.3 Enforcement Responsibilities

Community leaders are responsible for clarifying and enforcing our standards of acceptable behavior and will take appropriate and fair corrective action in response to any behavior that they deem inappropriate, threatening, offensive, or harmful.

Community leaders have the right and responsibility to remove, edit, or reject comments, commits, code, wiki edits, issues, and other contributions that are not aligned to this Code of Conduct, and will communicate reasons for moderation decisions when appropriate.

7.4 Scope

This Code of Conduct applies within all community spaces, and also applies when an individual is officially representing the community in public spaces. Examples of representing our community include using an official e-mail address, posting via an official social media account, or acting as an appointed representative at an online or offline event.

7.5 Enforcement

Instances of abusive, harassing, or otherwise unacceptable behavior may be reported to the community leaders responsible for enforcement at security@libranet.eu. All complaints will be reviewed and investigated promptly and fairly.

All community leaders are obligated to respect the privacy and security of the reporter of any incident.

7.6 Enforcement Guidelines

Community leaders will follow these Community Impact Guidelines in determining the consequences for any action they deem in violation of this Code of Conduct:

1. Correction

Community Impact: Use of inappropriate language or other behavior deemed unprofessional or unwelcome in the community.

Consequence: A private, written warning from community leaders, providing clarity around the nature of the violation and an explanation of why the behavior was inappropriate. A public apology may be requested.

2. Warning

Community Impact: A violation through a single incident or series of actions.

Consequence: A warning with consequences for continued behavior. No interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, for a specified period of time. This includes avoiding interactions in community spaces as well as external channels like social media. Violating these terms may lead to a temporary or permanent ban.

3. Temporary Ban

Community Impact: A serious violation of community standards, including sustained inappropriate behavior.

Consequence: A temporary ban from any sort of interaction or public communication with the community for a specified period of time. No public or private interaction with the people involved, including unsolicited interaction with those enforcing the Code of Conduct, is allowed during this period. Violating these terms may lead to a permanent ban.

4. Permanent Ban

Community Impact: Demonstrating a pattern of violation of community standards, including sustained inappropriate behavior, harassment of an individual, or aggression toward or disparagement of classes of individuals.

Consequence: A permanent ban from any sort of public interaction within the community.

7.7 Attribution

This Code of Conduct is adapted from the Contributor Covenant, version 2.0, available at https://www.contributor-covenant.org/version/2/0/code_of_conduct.html.

Community Impact Guidelines were inspired by Mozilla's code of conduct enforcement ladder.

For answers to common questions about this code of conduct, see the FAQ at https://www.contributor-covenant.org/faq. Translations are available at https://www.contributor-covenant.org/translations.

8 sitecustomize

sitecustomize.

8.1 Subpackages

sitecustomize._vendor

Subpackages

sitecustomize._vendor.importlib_metadata

Submodules

sitecustomize._vendor.importlib_metadata._adapters

Module Contents

Classes

Message	Basic message object.

class sitecustomize._vendor.importlib_metadata._adapters.**Message**(**args*, ***kwargs*)

Bases: email.message.Message

Basic message object.

A message object is defined as something that has a bunch of RFC 2822 headers and a payload. It may optionally have an envelope header (a.k.a. Unix-From or From_ header). If the message is a container (i.e. a multipart or a message/rfc822), then the payload is a list of Message objects, otherwise it is a string.

Message objects implement part of the `mapping' interface, which assumes there is exactly one occurrence of the header per message. Some headers do in fact appear multiple times (e.g. Received) and for those headers, you must use the explicit API to set or get all the headers. Not all of the mapping methods are implemented.

property json

Convert PackageMetadata to a JSON-compatible format per PEP 0566.

multiple_use_keys

Keys that may be indicated multiple times per PEP 566.

__iter__()

_repair_headers()

sitecustomize._vendor.importlib_metadata._collections

Module Contents

Classes

FreezableDefaultDict	Often it is desirable to prevent the mutation of
Pair	Built-in immutable sequence.

class sitecustomize._vendor.importlib_metadata._collections.**FreezableDefaultDict**

Bases: collections.defaultdict

Often it is desirable to prevent the mutation of a default dict after its initial construction, such as to prevent mutation during iteration.

```
>>> dd = FreezableDefaultDict(list)
>>> dd[0].append('1')
>>> dd.freeze()
>>> dd[1]
[]
>>> len(dd)
1
```

__missing__(*key*)

freeze()

class sitecustomize._vendor.importlib_metadata._collections.**Pair**

Bases: collections.namedtuple('Pair', 'name value')

Built-in immutable sequence.

If no argument is given, the constructor returns an empty tuple. If iterable is specified the tuple is initialized from iterable's items.

If the argument is a tuple, the return value is the same object.

classmethod parse(*text*)

sitecustomize._vendor.importlib_metadata._compat

Module Contents

Classes

NullFinder	A "Finder" (aka "MetaClassFinder") that never finds any modules,

Functions

install(cls)	Class decorator for installation on sys.meta_path.

sitecustomize._vendor.importlib_metadata._compat.**install**(*cls*)

Class decorator for installation on sys.meta_path.

Adds the backport DistributionFinder to sys.meta_path and attempts to disable the finder functionality of the stdlib DistributionFinder.

class sitecustomize._vendor.importlib_metadata._compat.**NullFinder**

A “Finder” (aka “MetaClassFinder”) that never finds any modules, but may find distributions.

find_module

static find_spec(**args*, ***kwargs*)

sitecustomize._vendor.importlib_metadata._functools

Module Contents

Functions

method_cache(method[, cache_wrapper])	Wrap lru_cache to support storing the cache data in the object instances.
pass_none(func)	Wrap func so it's not called if its first param is None

sitecustomize._vendor.importlib_metadata._functools.**method_cache**(*method*, *cache_wrapper=None*)

Wrap lru_cache to support storing the cache data in the object instances.

Abstracts the common paradigm where the method explicitly saves an underscore-prefixed protected property on first call and returns that subsequently.

```
>>> class MyClass:
...     calls = 0
...
...     @method_cache
...     def method(self, value):
```

(continues on next page)

```
...         self.calls += 1
...         return value
```

```
>>> a = MyClass()
>>> a.method(3)
3
>>> for x in range(75):
...     res = a.method(x)
>>> a.calls
75
```

Note that the apparent behavior will be exactly like that of lru_cache except that the cache is stored on each instance, so values in one instance will not flush values from another, and when an instance is deleted, so are the cached values for that instance.

```
>>> b = MyClass()
>>> for x in range(35):
...     res = b.method(x)
>>> b.calls
35
>>> a.method(0)
0
>>> a.calls
75
```

Note that if method had been decorated with `functools.lru_cache()`, a.calls would have been 76 (due to the cached value of 0 having been flushed by the 'b' instance).

Clear the cache with `.cache_clear()`

```
>>> a.method.cache_clear()
```

Same for a method that hasn't yet been called.

```
>>> c = MyClass()
>>> c.method.cache_clear()
```

Another cache wrapper may be supplied:

```
>>> cache = functools.lru_cache(maxsize=2)
>>> MyClass.method2 = method_cache(lambda self: 3, cache_wrapper=cache)
>>> a = MyClass()
>>> a.method2()
3
```

Caution - do not subsequently wrap the method with another decorator, such as `@property`, which changes the semantics of the function.

See also http://code.activestate.com/recipes/577452-a-memoize-decorator-for-instance-methods/ for another implementation and additional justification.

`sitecustomize._vendor.importlib_metadata._functools.`**`pass_none`**(*func*)

Wrap func so it's not called if its first param is None

```
>>> print_text = pass_none(print)
>>> print_text('text')
text
>>> print_text(None)
```

sitecustomize._vendor.importlib_metadata._itertools

Module Contents

Functions

unique_everseen(iterable[, key])	List unique elements, preserving order. Remember all elements ever seen.
always_iterable(obj[, base_type])	If *obj* is iterable, return an iterator over its items:

sitecustomize._vendor.importlib_metadata._itertools.**unique_everseen**(*iterable*, *key=None*)

List unique elements, preserving order. Remember all elements ever seen.

sitecustomize._vendor.importlib_metadata._itertools.**always_iterable**(*obj*, *base_type=(str, bytes)*)

If *obj* is iterable, return an iterator over its items:

```
>>> obj = (1, 2, 3)
>>> list(always_iterable(obj))
[1, 2, 3]
```

If *obj* is not iterable, return a one-item iterable containing *obj*:

```
>>> obj = 1
>>> list(always_iterable(obj))
[1]
```

If *obj* is `None`, return an empty iterable:

```
>>> obj = None
>>> list(always_iterable(None))
[]
```

By default, binary and text strings are not considered iterable:

```
>>> obj = 'foo'
>>> list(always_iterable(obj))
['foo']
```

If *base_type* is set, objects for which `isinstance(obj, base_type)` returns `True` won't be considered iterable.

```
>>> obj = {'a': 1}
>>> list(always_iterable(obj))  # Iterate over the dict's keys
['a']
>>> list(always_iterable(obj, base_type=dict))  # Treat dicts as a unit
[{'a': 1}]
```

Set *base_type* to `None` to avoid any special handling and treat objects Python considers iterable as iterable:

```
>>> obj = 'foo'
>>> list(always_iterable(obj, base_type=None))
['f', 'o', 'o']
```

sitecustomize._vendor.importlib_metadata._meta

Module Contents

Classes

PackageMetadata	Base class for protocol classes.
SimplePath	A minimal subset of pathlib.Path required by PathDistribution.

Attributes

_T

sitecustomize._vendor.importlib_metadata._meta._T

class sitecustomize._vendor.importlib_metadata._meta.**PackageMetadata**

Bases: sitecustomize._vendor.importlib_metadata._compat.Protocol

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing), for example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

property json: Dict[str, str | List[str]]

A JSON-compatible form of the metadata.

Return type
Dict[str, Union[str, List[str]]]

__len__()

Return type
int

__contains__(*item*)

Parameters
item (*str*) –

Return type
bool

__getitem__(*key*)

Parameters
key (*str*) –

Return type
str

__iter__()

Return type
Iterator[str]

get_all(*name*, *failobj=...*)

Return all values associated with a possibly multi-valued key.

Parameters

- **name** (*str*) –
- **failobj** (*_T*) –

Return type
Union[List[Any], _T]

class sitecustomize._vendor.importlib_metadata._meta.**SimplePath**

Bases: sitecustomize._vendor.importlib_metadata._compat.Protocol

A minimal subset of pathlib.Path required by PathDistribution.

joinpath()

Return type
SimplePath

__truediv__()

Return type
SimplePath

parent()

Return type
SimplePath

read_text()

Return type
str

sitecustomize._vendor.importlib_metadata._text

Module Contents

Classes

FoldedCase	A case insensitive string class; behaves just like str

class sitecustomize._vendor.importlib_metadata._text.**FoldedCase**

Bases: str

A case insensitive string class; behaves just like str except compares equal when the only variation is case.

```
>>> s = FoldedCase('hello world')
```

```
>>> s == 'Hello World'
True
```

```
>>> 'Hello World' == s
True
```

```
>>> s != 'Hello World'
False
```

```
>>> s.index('O')
4
```

```
>>> s.split('O')
['hell', ' w', 'rld']
```

```
>>> sorted(map(FoldedCase, ['GAMMA', 'alpha', 'Beta']))
['alpha', 'Beta', 'GAMMA']
```

Sequence membership is straightforward.

```
>>> "Hello World" in [s]
True
>>> s in ["Hello World"]
True
```

You may test for set inclusion, but candidate and elements must both be folded.

```
>>> FoldedCase("Hello World") in {s}
True
>>> s in {FoldedCase("Hello World")}
True
```

String inclusion works as long as the FoldedCase object is on the right.

```
>>> "hello" in FoldedCase("Hello World")
True
```

But not if the FoldedCase object is on the left:

```
>>> FoldedCase('hello') in 'Hello World'
False
```

In that case, use **in_**:

```
>>> FoldedCase('hello').in_('Hello World')
True
```

```
>>> FoldedCase('hello') > FoldedCase('Hello')
False
```

__lt__(*other*)

Return self<value.

__gt__(*other*)

Return self>value.

__eq__(*other*)

Return self==value.

__ne__(*other*)

Return self!=value.

__hash__()

Return hash(self).

__contains__(*other*)

Return key in self.

in_(*other*)

Does self appear in other?

lower()

Return a copy of the string converted to lowercase.

index(*sub*)

S.index(sub[, start[, end]]) -> int

Return the lowest index in S where substring sub is found, such that sub is contained within S[start:end]. Optional arguments start and end are interpreted as in slice notation.

Raises ValueError when the substring is not found.

split(*splitter=' '*, *maxsplit=0*)

Return a list of the substrings in the string, using sep as the separator string.

sep

The separator used to split the string.

When set to None (the default value), will split on any whitespace character (including \n \r \t \f and spaces) and will discard empty strings from the result.

maxsplit

Maximum number of splits (starting from the left). -1 (the default value) means no limit.

Note, str.split() is mainly useful for data that has been intentionally delimited. With natural text that includes punctuation, consider using the regular expression module.

Package Contents

Classes

PackageMetadata	Base class for protocol classes.
Distribution	A Python distribution package.
DistributionFinder	A MetaPathFinder capable of discovering installed distributions.

Functions

distribution(distribution_name)	Get the `Distribution` instance for the named package.
distributions(**kwargs)	Get all `Distribution` instances in the current environment.
metadata(distribution_name)	Get the metadata for the named package.
version(distribution_name)	Get the version string for the named package.
entry_points(**params)	Return EntryPoint objects for all installed packages.
files(distribution_name)	Return a list of files for the named package.
requires(distribution_name)	Return a list of requirements for the named package.
packages_distributions()	Return a mapping of top-level packages to their

class `sitecustomize._vendor.importlib_metadata.`**PackageMetadata**

Bases: `sitecustomize._vendor.importlib_metadata._compat.Protocol`

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing), for example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

property json: Dict[str, str | List[str]]

A JSON-compatible form of the metadata.

Return type
Dict[str, Union[str, List[str]]]

__len__()

Return type
int

__contains__(*item*)

Parameters
item (*str*) –

Return type
bool

__getitem__(*key*)

Parameters
key (*str*) –

Return type
str

__iter__()

Return type
Iterator[str]

get_all(*name*, *failobj=...*)

Return all values associated with a possibly multi-valued key.

Parameters

- **name** (*str*) –
- **failobj** (*_T*) –

Return type
Union[List[Any], _T]

exception sitecustomize._vendor.importlib_metadata.**PackageNotFoundError**

Bases: ModuleNotFoundError

The package was not found.

property name

__str__()

Return str(self).

class sitecustomize._vendor.importlib_metadata.**Distribution**

A Python distribution package.

property metadata: *_meta.PackageMetadata*

Return the parsed metadata for this Distribution.

The returned object will have keys that name the various bits of metadata. See PEP 566 for details.

Return type
_meta.PackageMetadata

property name

Return the 'Name' metadata for the distribution package.

property _normalized_name

Return a normalized version of the name.

property version

Return the 'Version' metadata for the distribution package.

property entry_points

property files

Files in this distribution.

Returns
List of PackagePath for this distribution or None

Result is *None* if the metadata file that enumerates files (i.e. RECORD for dist-info or SOURCES.txt for egg-info) is missing. Result may be empty if the metadata exists but is empty.

property requires

Generated requirements specified for this Distribution

abstract read_text(*filename*)

Attempt to load metadata file given by the name.

Parameters
filename – The name of the file in the distribution info.

Returns
The text if found, otherwise None.

abstract locate_file(*path*)

Given a path to a file in this distribution, return a path to it.

classmethod from_name(*name*)

Return the Distribution for the given package name.

Parameters
name – The name of the distribution package to search for.

Returns
The Distribution instance (or subclass thereof) for the named package, if found.

Raises
PackageNotFoundError – When the named package's distribution metadata cannot be found.

classmethod discover(***kwargs*)

Return an iterable of Distribution objects for all packages.

Pass a `context` or pass keyword arguments for constructing a context.

Context
A `DistributionFinder.Context` object.

Returns
Iterable of Distribution objects for all packages.

static at(*path*)

Return a Distribution for the indicated metadata path

Parameters
path – a string or path-like object

Returns
a concrete Distribution instance for the path

static _discover_resolvers()

Search the meta_path for resolvers.

_read_files_distinfo()

Read the lines of RECORD

_read_files_egginfo()

SOURCES.txt might contain literal commas, so wrap each line in quotes.

_read_dist_info_reqs()

_read_egg_info_reqs()

classmethod _deps_from_requires_text(*source*)

static _convert_egg_info_reqs_to_simple_reqs(*sections*)

Historically, setuptools would solicit and store 'extra' requirements, including those with environment markers, in separate sections. More modern tools expect each dependency to be defined separately, with any relevant extras and environment markers attached directly to that requirement. This method converts the former to the latter. See _test_deps_from_requires_text for an example.

class sitecustomize._vendor.importlib_metadata.**DistributionFinder**

Bases: importlib.abc.MetaPathFinder

A MetaPathFinder capable of discovering installed distributions.

class Context(***kwargs*)

Keyword arguments presented by the caller to distributions() or Distribution.discover() to narrow the scope of a search for distributions in all DistributionFinders.

Each DistributionFinder may expect any parameters and should attempt to honor the canonical parameters defined below when appropriate.

property path

The sequence of directory path that a distribution finder should search.

Typically refers to Python installed package paths such as "site-packages" directories and defaults to sys.path.

name

Specific name for which a distribution finder should match. A name of None matches all distributions.

abstract find_distributions(*context=Context()*)

Find distributions.

Return an iterable of all Distribution instances capable of loading the metadata for packages matching the context, a DistributionFinder.Context instance.

sitecustomize._vendor.importlib_metadata.**distribution**(*distribution_name*)

Get the Distribution instance for the named package.

Parameters
distribution_name – The name of the distribution package as a string.

Returns
A Distribution instance (or subclass thereof).

sitecustomize._vendor.importlib_metadata.**distributions**(***kwargs*)

Get all Distribution instances in the current environment.

Returns
An iterable of Distribution instances.

sitecustomize._vendor.importlib_metadata.**metadata**(*distribution_name*)

Get the metadata for the named package.

Parameters
distribution_name – The name of the distribution package to query.

Returns
A PackageMetadata containing the parsed metadata.

Return type
_meta.PackageMetadata

sitecustomize._vendor.importlib_metadata.**version**(*distribution_name*)

Get the version string for the named package.

Parameters
distribution_name – The name of the distribution package to query.

Returns
The version string for the package as defined in the package's "Version" metadata key.

sitecustomize._vendor.importlib_metadata.**entry_points**(***params*)

Return EntryPoint objects for all installed packages.

Pass selection parameters (group or name) to filter the result to entry points matching those properties (see EntryPoints.select()).

For compatibility, returns `SelectableGroups` object unless selection parameters are supplied. In the future, this function will return `EntryPoints` instead of `SelectableGroups` even when no selection parameters are supplied.

For maximum future compatibility, pass selection parameters or invoke `.select` with parameters on the result.

Returns
EntryPoints or SelectableGroups for all installed packages.

Return type
Union[EntryPoints, SelectableGroups]

sitecustomize._vendor.importlib_metadata.**files**(*distribution_name*)

Return a list of files for the named package.

Parameters
distribution_name – The name of the distribution package to query.

Returns
List of files composing the distribution.

sitecustomize._vendor.importlib_metadata.**requires**(*distribution_name*)

Return a list of requirements for the named package.

Returns
An iterator of requirements, suitable for packaging.requirement.Requirement.

sitecustomize._vendor.importlib_metadata.**packages_distributions**()

Return a mapping of top-level packages to their distributions.

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

Return type
Mapping[str, List[str]]

Submodules

sitecustomize._vendor.zipp

Module Contents

Classes

Path	A pathlib-compatible interface for zip files.

class sitecustomize._vendor.zipp.**Path**(*root*, *at=''*)

A pathlib-compatible interface for zip files.

Consider a zip file with this structure:

```
.
├── a.txt
└── b
    ├── c.txt
    └── d
        └── e.txt
```

```
>>> data = io.BytesIO()
>>> zf = zipfile.ZipFile(data, 'w')
>>> zf.writestr('a.txt', 'content of a')
>>> zf.writestr('b/c.txt', 'content of c')
>>> zf.writestr('b/d/e.txt', 'content of e')
>>> zf.filename = 'mem/abcde.zip'
```

Path accepts the zipfile object itself or a filename

```
>>> root = Path(zf)
```

From there, several path operations are available.

Directory iteration (including the zip file itself):

```
>>> a, b = root.iterdir()
>>> a
Path('mem/abcde.zip', 'a.txt')
>>> b
Path('mem/abcde.zip', 'b/')
```

name property:

```
>>> b.name
'b'
```

join with divide operator:

```
>>> c = b / 'c.txt'
>>> c
Path('mem/abcde.zip', 'b/c.txt')
>>> c.name
'c.txt'
```

Read text:

```
>>> c.read_text()
'content of c'
```

existence:

```
>>> c.exists()
True
>>> (b / 'missing.txt').exists()
False
```

Coercion to string:

```
>>> import os
>>> str(c).replace(os.sep, posixpath.sep)
'mem/abcde.zip/b/c.txt'
```

At the root, `name`, `filename`, and `parent` resolve to the zipfile. Note these attributes are not valid and will raise a `ValueError` if the zipfile has no filename.

```
>>> root.name
'abcde.zip'
>>> str(root.filename).replace(os.sep, posixpath.sep)
'mem/abcde.zip'
>>> str(root.parent)
'mem'
```

property name

property suffix

property suffixes

property stem

property filename

property parent

__repr = '{self.__class__.__name__}({self.root.filename!r}, {self.at!r})'

__truediv__

open(*mode='r'*, **args*, *pwd=None*, ***kwargs*)

Open this entry as text or binary following the semantics of `pathlib.Path.open()` by passing arguments through to io.TextIOWrapper().

read_text(**args*, ***kwargs*)

read_bytes()

_is_child(*path*)

_next(*at*)

is_dir()

is_file()

exists()

iterdir()

__str__()
Return str(self).

__repr__()
Return repr(self).

joinpath(**other*)

8.2 Submodules

sitecustomize._utils

sitecustomize._utils.

Module Contents

Classes

NamedObject	Base class for protocol classes.
SimpleWarning	

Functions

fifo_filter(ordered_list)	Remove duplicate entries from an ordered list,

class sitecustomize._utils.**NamedObject**
Bases: Protocol

Base class for protocol classes.

Protocol classes are defined as:

```
class Proto(Protocol):
    def meth(self) -> int:
        ...
```

Such classes are primarily used with static type checkers that recognize structural subtyping (static duck-typing), for example:

```
class C:
    def meth(self) -> int:
        return 0

def func(x: Proto) -> int:
    return x.meth()

func(C())  # Passes static type check
```

See PEP 544 for details. Protocol classes decorated with @typing.runtime_checkable act as simple-minded runtime protocols that check only the presence of given attributes, ignoring their type signatures. Protocol classes can be generic, they are defined as:

```
class GenProto(Protocol[T]):
    def meth(self) -> T:
        ...
```

name: str

sitecustomize._utils.**fifo_filter**(*ordered_list*)

Remove duplicate entries from an ordered list, preserving initial ordering but removing previously seen entries.

Example

```
>>> fifo_filter([1, 2, 3, 1])
  [2, 3, 1]
```

This allows you to override the ordering of a registered entrypoint in your own pyproject.toml.

Parameters
ordered_list (*List[NamedObject]*) –

Return type
List[*NamedObject*]

class sitecustomize._utils.**SimpleWarning**

__enter__()

__exit__(**args*)

static simple_warning_format(*msg*, *category*, *filename*, *lineno*, *file=None*, *line=None*)

Simple warning-formatting.

8.3 Package Contents

Classes

SimpleWarning

Functions

fifo_filter(ordered_list)	Remove duplicate entries from an ordered list,
entry_points(**params)	Return EntryPoint objects for all installed packages.
cancel()	No-op function to cancel registered entrypoints.
print_entrypoints([group_name, filtered])	print registered entrypoints.

Attributes

__version__
ENTRYPOINT_GROUPNAME
eps

sitecustomize.**__version__ = '1.0.0'**

class sitecustomize.**SimpleWarning**

__enter__()

__exit__(**args*)

static simple_warning_format(*msg*, *category*, *filename*, *lineno*, *file=None*, *line=None*)

Simple warning-formatting.

sitecustomize.**fifo_filter**(*ordered_list*)

Remove duplicate entries from an ordered list, preserving initial ordering but removing previously seen entries.

Example

```
>>> fifo_filter([1, 2, 3, 1])
  [2, 3, 1]
```

This allows you to override the ordering of a registered entrypoint in your own pyproject.toml.

Parameters
ordered_list (*List[NamedObject]*) –

Return type
List[*NamedObject*]

sitecustomize.**entry_points**(***params*)

Return EntryPoint objects for all installed packages.

Pass selection parameters (group or name) to filter the result to entry points matching those properties (see EntryPoints.select()).

For compatibility, returns `SelectableGroups` object unless selection parameters are supplied. In the future, this function will return `EntryPoints` instead of `SelectableGroups` even when no selection parameters are supplied.

For maximum future compatibility, pass selection parameters or invoke `.select` with parameters on the result.

Returns
EntryPoints or SelectableGroups for all installed packages.

Return type
Union[EntryPoints, SelectableGroups]

sitecustomize.**ENTRYPOINT_GROUPNAME = 'sitecustomize'**

sitecustomize.**eps**

sitecustomize.**cancel**()

No-op function to cancel registered entrypoints.

Imagine your project depends on a package that registers a sitecustomize-entrypoint:

In third third-party package:
[tool.poetry.plugins."sitecustomize"] foo = "package_foo:foo"

And you register a sitecustomize-entrypoint in your own project:
[tool.poetry.plugins."sitecustomize"] bar = "package_bar:bar"

You can guarantee the ordering, bv canceling the registered entrypoints and re-registering them in the order you want:

in our own pyproject.toml:

[tool.poetry.plugins."sitecustomize"] foo = "sitecustomize:cancel" bar = "sitecustomize:cancel"

1_bar = "package_bar:bar" 2_foo = "package_foo:foo"

But please becautioned that the ordering in your pyproject.toml is irrelevant and that after parsing the names of entrypoints will be sorted alpanumerically. Therefore we advice to use integer-prefixes.

Return type
None

sitecustomize.**print_entrypoints**(*group_name=ENTRYPOINT_GROUPNAME*, *filtered=False*)

print registered entrypoints.

Parameters

- **group_name** (*str*) –
- **filtered** (*bool*) –

9 Indices and tables

- genindex
- modindex
- search

Python Module Index

S

Index

Symbols

A

C

N

O

P

R

S

U

V