shippinglabel

Release 1.7.1

Utilities for handling packages.

Dominic Davis-Foster

Jan 10, 2024

Contents

Chapter

ONE

Installation

1.1 from PyPI

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

1.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install shippinglabel
```

1.3 from GitHub

```
$ python3 -m pip install git+https://github.com/domdfcoding/shippinglabel@master --user
```

`shippinglabel` includes a vendored copy of trove-classifiers. If you install a newer version of `trove-classifiers` with pip `shippinglabel` will use that version instead.

Chapter

TWO

shippinglabel

Utilities for handling packages.

Functions:

get_project_links(project_name[, path])	Returns the web links for the given project.
no_dev_versions(versions)	Returns the subset of `versions` which does not end with `-dev`.
no_pre_versions(versions)	Returns the subset of `versions` which are not prereleases (alpha, beta, dev, rc etc.).
normalize(name)	Normalize the given name for PyPI et al.
normalize_keep_dot(name)	Normalize the given name for PyPI et al., but keep dots in namespace packages.
read_pyvenv(venv_dir)	Reads the `pyvenv.cfg` for the given virtualenv, and returns a `key: value` mapping of its contents.

get_project_links (*project_name*, *path=None*)

Returns the web links for the given project.

The exact keys vary, but common keys include “Documentation” and “Issue Tracker”.

New in version 0.12.0.

Parameters

- **project_name** (`str`)
- **path** (`Optional`[`Iterable`[`Union`[`str`, `Path`, `PathLike`]]]) – The directories entries to search for distributions in. This can be used to search in a different (virtual) environment. Default `sys.path`.

Return type `MetadataMapping`

Changed in version 1.0.0: Now returns a `dist_meta.metadata_mapping.MetadataMapping` object.

Changed in version 1.2.0: The Home-Page field from Python core metadata is included under the `Homepage` key, if present. This matches the output parsed from PyPI for packages which are not installed.

Changed in version 1.7.0: Added the `path` argument.

no_dev_versions (*versions*)

Returns the subset of `versions` which does not end with `-dev`.

Parameters **versions** (`Iterable`[`str`])

Return type `List`[`str`]

no_pre_versions(*versions*)
Returns the subset of `versions` which are not prereleases (alpha, beta, dev, rc etc.).

New in version 0.15.0.

Parameters **versions** (`Iterable[str]`)

Return type `List[str]`

normalize(*name*)
Normalize the given name for PyPI et al.

From **PEP 503** (public domain).

Parameters **name** (`str`) – The project name.

Return type `str`

normalize_keep_dot(*name*)
Normalize the given name for PyPI et al., but keep dots in namespace packages.

New in version 0.2.1.

Parameters **name** (`str`) – The project name.

Return type `str`

read_pyvenv(*venv_dir*)
Reads the `pyvenv.cfg` for the given virtualenv, and returns a `key: value` mapping of its contents.

New in version 0.9.0.

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

Return type `Dict[str, str]`

Chapter

THREE

shippinglabel.checksum

Utilities for creating and checking file sha256 checksums.

New in version 0.6.0.

Functions:

get_sha256_hash(filename[, blocksize])	Returns the SHA256 hash object for the given file.
check_sha256_hash(filename, hash[, blocksize])	Returns whether the SHA256 hash for the file matches `hash`.
get_md5_hash(filename[, blocksize])	Returns the md5 hash object for the given file.
get_record_entry(filename[, blocksize, ...])	Constructs a **PEP 376** `RECORD` entry for the file.

get_sha256_hash (*filename*, *blocksize=1048576*)

Returns the SHA256 hash object for the given file.

New in version 0.6.0.

Parameters

- **filename** (`Union`[`str`, `Path`, `PathLike`, `IO`[`bytes`]])
- **blocksize** (`int`) – The blocksize to read the file with. Default `1048576`.

Return type `hashlib.sha256()`

Changed in version 0.16.0: Added support for already open file objects.

check_sha256_hash (*filename*, *hash*, *blocksize=1048576*)

Returns whether the SHA256 hash for the file matches `hash`.

New in version 0.6.0.

Parameters

- **filename** (`Union`[`str`, `Path`, `PathLike`, `IO`[`bytes`]])
- **hash** (`Union`[`hashlib.sha256()`, `str`]) – If a string, the hexdigest of the hash.
- **blocksize** (`int`) – The blocksize to read the file with. Default `1048576`.

Return type `bool`

Changed in version 0.16.0: Added support for already open file objects.

get_md5_hash (*filename*, *blocksize=1048576*)
Returns the md5 hash object for the given file.

New in version 0.15.0.

Parameters

- **filename** (Union[str, Path, PathLike, IO[bytes]])
- **blocksize** (int) – The blocksize to read the file with. Default 1048576.

Return type hashlib.md5()

Changed in version 0.16.0: Added support for already open file objects.

get_record_entry (*filename*, *blocksize=1048576*, *relative_to=None*)
Constructs a **PEP 376** RECORD entry for the file.

New in version 0.6.0.

Parameters

- **filename** (Union[str, Path, PathLike])
- **blocksize** (int) – The blocksize to read the file with. Default 1048576.
- **relative_to** (Union[str, Path, PathLike, None]) – Default None.

Return type str

Chapter

FOUR

shippinglabel.classifiers

Utilities for working with trove classifiers.

Functions:

classifiers_from_requirements(requirements)	Returns an iterator over suggested trove classifiers based on the given requirements.
validate_classifiers(classifiers)	Validate a list of trove classifiers.

classifiers_from_requirements(*requirements*)

Returns an iterator over suggested trove classifiers based on the given requirements.

New in version 0.5.0.

Parameters **requirements** (Collection[*ComparableRequirement*])

Return type Iterator[str]

validate_classifiers(*classifiers*)

Validate a list of trove classifiers.

Parameters **classifiers** (Iterable[str])

Return type bool

Chapter

FIVE

shippinglabel.conda

Attention: This module is deprecated and will be removed in v2.0.0. Please use the new *shippinglabel_conda* extension instead.

Functions to aid building of conda packages.

New in version 0.7.0.

Data:

CONDA_API	Instance of apeye.slumber_url.SlumberURL for accessing the Conda API.

Functions:

clear_cache(*channel_name)	Clear the cached Conda channel listings.
compile_requirements(repo_dir[, extras])	Compile a list of requirements for the package from the requirements.txt file, and any extra dependencies.
get_channel_listing(channel_name)	Obtain the list of packages in the given Conda channel, either from the cache or from the Conda API.
make_conda_description(summary[, conda_channels])	Create a description for the Conda package from its summary and a list of channels required to install it.
prepare_requirements(requirements)	Prepare a list of requirements for use with conda.
validate_requirements(requirements, ...)	Ensure that all requirements are available from the given Conda channels, and normalize the names to those in the Conda channel.

CONDA_API = SlumberURL('https://conda.anaconda.org')

Type: SlumberURL

Instance of apeye.slumber_url.SlumberURL for accessing the Conda API.

New in version 0.7.0.

clear_cache(**channel_name*)

Clear the cached Conda channel listings.

New in version 0.7.0.

Parameters ***channel_name** (str) – The name(s) of the channels to clear the cache for.

If no arguments are given the cache is cleared for all channels.

compile_requirements(*repo_dir*, *extras=()*)

Compile a list of requirements for the package from the `requirements.txt` file, and any extra dependencies.

New in version 0.7.0.

Parameters

- **repo_dir** (PathPlus)
- **extras** (Iterable[str]) – A list of additional, optional requirements. These would be specified in "extras_require" for setuptools. Default `()`.

Return type List[*ComparableRequirement*]

get_channel_listing(*channel_name*)

Obtain the list of packages in the given Conda channel, either from the cache or from the Conda API.

Responses are cached for 48 hours. The cache can be cleared manually with *clear_cache()*.

New in version 0.7.0.

Parameters **channel_name** (str)

Raises **ValueError** – if the channel can't be found (*New in version 0.15.0*).

Return type List[str]

make_conda_description(*summary*, *conda_channels=()*)

Create a description for the Conda package from its summary and a list of channels required to install it.

The description will look like:

```
This is my fancy Conda package. Hope you like it ☺.


Before installing please ensure you have added the following channels: conda-
↪forge, bioconda
```

if called as follows:

```
make_conda_description(
    "This is my fancy Conda package. Hope you like it ☺.",
    ["conda-forge", "bioconda"],
)
```

New in version 0.8.0.

Parameters

- **summary** (str)
- **conda_channels** (Iterable[str]) – Default `()`.

Return type str

prepare_requirements(*requirements*)

Prepare a list of requirements for use with conda.

This entails removing any extras and markers from the requirements, and skipping any requirements with URLs, as conda does not support these.

New in version 0.13.0.

Parameters **requirements** (Iterable[*ComparableRequirement*])

Return type Iterable[*ComparableRequirement*]

validate_requirements(*requirements*, *conda_channels*)

Ensure that all requirements are available from the given Conda channels, and normalize the names to those in the Conda channel.

New in version 0.7.0.

Parameters

- **requirements** (Iterable[*ComparableRequirement*])
- **conda_channels** (Iterable[str])

Return type List[*ComparableRequirement*]

Chapter

SIX

shippinglabel.pypi

Attention: This module is deprecated and will be removed in v2.0.0. Please use the new *shippinglabel_pypi* extension instead.

Utilities for working with the Python Package Index (PyPI).

New in version 0.2.0.

See also:

pypi-json, which provides some of the functionality from this module but with a reusable HTTP session and support for authentication with other endpoints (such as a private package repository).

Classes:

FileURL	`typing.TypedDict` representing the output of *get_releases_with_digests()*.

Data:

PYPI_API	Instance of `apeye.slumber_url.SlumberURL` which points to the PyPI REST API.

Functions:

bind_requirements(filename[, specifier, ...])	Bind unbound requirements in the given file to the latest version on PyPI, and any later versions.
get_file_from_pypi(url, tmpdir)	Download the file with the given URL into the given (temporary) directory.
get_latest(pypi_name)	Returns the version number of the latest release on PyPI for the given project.
get_metadata(pypi_name)	Returns metadata for the given project on PyPI.
get_pypi_releases(pypi_name)	Returns a dictionary mapping PyPI release versions to download URLs.
get_releases_with_digests(pypi_name)	Returns a dictionary mapping PyPI release versions to download URLs and the sha256sum of the file contents.
get_sdist_url(name, version[, strict])	Returns the URL of the project's source distribution on PyPI.
get_wheel_tag_mapping(name, version)	Constructs a mapping of wheel tags to the PyPI download URL of the wheel with relevant tag.
get_wheel_url(name, version[, strict])	Returns the URL of one of the project's wheels on PyPI.

typeddict FileURL
Bases: TypedDict

typing.TypedDict representing the output of *get_releases_with_digests()*.

New in version 0.6.1.

Required Keys

- **url** (str)
- **digest** (str)

PYPI_API = SlumberURL('https://pypi.org/pypi')
Type: SlumberURL

Instance of apeye.slumber_url.SlumberURL which points to the PyPI REST API.

Changed in version 0.3.0: Now an instance of apeye.slumber_url.SlumberURL.

bind_requirements (*filename*, *specifier='>='*, *normalize_func=<function 'normalize'>*)
Bind unbound requirements in the given file to the latest version on PyPI, and any later versions.

New in version 0.2.0.

Parameters

- **filename** (Union[str, Path, PathLike]) – The requirements.txt file to bind requirements in.
- **specifier** (str) – The requirement specifier symbol to use. Default '>='.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of requirements. Default *shippinglabel.normalize()*.

Changed in version 0.2.3: Added the normalize_func keyword-only argument.

Return type int

Returns 1 if the file was changed; 0 otherwise.

get_file_from_pypi (*url*, *tmpdir*)
Download the file with the given URL into the given (temporary) directory.

New in version 0.6.0.

Parameters

- **url** (Union[URL, str]) – The URL to download the file from.
- **tmpdir** (Union[str, Path, PathLike]) – The (temporary) directory to store the downloaded file in.

get_latest (*pypi_name*)
Returns the version number of the latest release on PyPI for the given project.

New in version 0.2.0.

Parameters **pypi_name** (str)

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.

- apeye.slumber_url.exceptions.HttpServerError if an error occurs when communicating with PyPI.

Return type str

get_metadata(*pypi_name*)
Returns metadata for the given project on PyPI.

New in version 0.2.0.

Parameters **pypi_name** (str)

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- apeye.slumber_url.exceptions.HttpServerError if an error occurs when communicating with PyPI.

Return type Dict[str, Any]

get_pypi_releases(*pypi_name*)
Returns a dictionary mapping PyPI release versions to download URLs.

New in version 0.3.0.

Parameters **pypi_name** (str) – The name of the project on PyPI.

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- apeye.slumber_url.exceptions.HttpServerError if an error occurs when communicating with PyPI.

Return type Dict[str, List[str]]

get_releases_with_digests(*pypi_name*)
Returns a dictionary mapping PyPI release versions to download URLs and the sha256sum of the file contents.

New in version 0.6.0.

Parameters **pypi_name** (str) – The name of the project on PyPI.

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- apeye.slumber_url.exceptions.HttpServerError if an error occurs when communicating with PyPI.

Return type Dict[str, List[*FileURL*]]

get_sdist_url(*name*, *version*, *strict=False*)
Returns the URL of the project's source distribution on PyPI.

New in version 0.13.0.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])

- **strict** (bool) – Causes a ValueError to be raised if no sdist is found, rather than retuning a wheel. Default False.

Attention: If no source distribution is found this function may return a wheel or "zip" sdist unless strict is True.

Changed in version 0.15.0: Added the strict argument.

Return type str

get_wheel_tag_mapping(*name*, *version*)

Constructs a mapping of wheel tags to the PyPI download URL of the wheel with relevant tag.

This can be used alongside packaging.tags.sys_tags() to select the best wheel for the current platform.

New in version 0.15.0.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])

Return type Tuple[Dict[Tag, URL], List[URL]]

Returns A tuple containing the tag: url mapping, and a list of download URLs for non-wheel artifacts (e.g. sdists).

get_wheel_url(*name*, *version*, *strict=False*)

Returns the URL of one of the project's wheels on PyPI.

For finer control over which wheel the URL is for see the *get_wheel_tag_mapping()* function.

New in version 0.15.0.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])
- **strict** (bool) – Causes a ValueError to be raised if no wheels are found, rather than retuning a wheel. Default False.

Attention: If no wheels are found this function may return an sdist unless strict is True.

Return type str

Chapter

SEVEN

shippinglabel.requirements

Utilities for working with **PEP 508** requirements.

Classes:

ComparableRequirement(requirement_string)	Represents a **PEP 508** requirement.
RequirementsManager(repo_path)	Abstract base class for managing requirements files.

Functions:

check_dependencies(dependencies[, prt])	Check whether one or more dependencies are available to be imported.
combine_requirements(requirement, *requirements)	Combine duplicated requirements in a list.
list_requirements(name[, depth, path])	Returns an iterator over the requirements of the given library, and the requirements of those requirements.
parse_pyproject_dependencies(pyproject_file)	Parse the project's dependencies from its `pyproject.toml` file.
parse_pyproject_extras(pyproject_file[, ...])	Parse the project's extra dependencies from its `pyproject.toml` file.
parse_requirements(requirements, *[, ...])	Parse the given strings as **PEP 508** requirements.
read_requirements(req_file[, ...])	Reads **PEP 508** requirements from the given file.
resolve_specifiers(specifiers)	Resolve duplicated and overlapping requirement specifiers.

class ComparableRequirement(*requirement_string*)

Bases: Requirement

Represents a **PEP 508** requirement.

Can be compared to other requirements. A list of *ComparableRequirement* objects can be sorted alphabetically.

Methods:

__eq__(other)	Return `self == other`.
__ge__(other)	Return `self >= other`.
__gt__(other)	Return `self > other`.
__le__(other)	Return `self <= other`.
__lt__(other)	Return `self < other`.

__eq__(*other*)
Return `self == other`.

Return type bool

__ge__(*other*)
Return `self >= other`.

Return type bool

__gt__(*other*)
Return `self > other`.

Return type bool

__le__(*other*)
Return `self <= other`.

Return type bool

__lt__(*other*)
Return `self < other`.

Return type bool

class RequirementsManager(*repo_path*)
Bases: ABC

Abstract base class for managing requirements files.

When invoked with run, the methods are called in the following order:

1. *compile_target_requirements()*
2. *merge_requirements()*
3. *remove_library_requirements()*
4. *write_requirements()*

Parameters **repo_path** (Union[str, Path, PathLike]) – Path to the repository root.

Methods:

compile_target_requirements()	Add and remove requirements depending on the configuration by modifying the `target_requirements` attribute.
get_target_requirement_names()	Returns a list of normalized names for the target requirements, including any added by `compile_target_requirements`.
merge_requirements()	Merge requirements already in the file with the target requirements.
normalize(name)	Normalize the given name for PyPI et al.
prep_req_file()	Create the requirements file if necessary, and in any case return its filename.

continues on next page

Table 4 – continued from previous page

remove_library_requirements()	Remove requirements given in the library's `requirements.txt` file.
run()	Update the list of requirements and return the name of the requirements file.
write_requirements(comments)	Write the list of requirements to the file.

Attributes:

filename	The path of the requirements file, relative to the repository root.
target_requirements	The static target requirements

compile_target_requirements()
Add and remove requirements depending on the configuration by modifying the `target_requirements` attribute.

This method may not return anything.

filename
Type: `Union[str, Path, PathLike]`

The path of the requirements file, relative to the repository root.

get_target_requirement_names()
Returns a list of normalized names for the target requirements, including any added by `compile_target_requirements`.

Return type `Set[str]`

merge_requirements()
Merge requirements already in the file with the target requirements.

Requirements may be added, changed or removed at this stage by modifying the `target_requirements` attribute.

Return type `List[str]`

Returns List of commented lines.

normalize(*name*)
Normalize the given name for PyPI et al.

New in version 0.2.1.

Parameters **name** (`str`) – The project name.

Return type `str`

prep_req_file()
Create the requirements file if necessary, and in any case return its filename.

Return type `PathPlus`

remove_library_requirements()
Remove requirements given in the library's `requirements.txt` file.

This method may not return anything.

run()
Update the list of requirements and return the name of the requirements file.

Return type PathPlus

target_requirements
Type: Set[*ComparableRequirement*]

The static target requirements

Changed in version 0.4.0: Previously this was a set of `packaging.requirements.Requirement`.

write_requirements(*comments*)
Write the list of requirements to the file.

Parameters **comments** (List[str]) – List of commented lines.

This method may not return anything.

check_dependencies(*dependencies*, *prt=True*)
Check whether one or more dependencies are available to be imported.

Parameters

- **dependencies** (Iterable[str]) – The list of dependencies to check the availability of.
- **prt** (bool) – Whether the status should be printed to the terminal. Default True.

Return type List[str]

Returns A list of any missing modules.

Deprecated since version 1.6.0: This will be removed in 2.0.

combine_requirements(*requirement*, **requirements*, *normalize_func=<function 'normalize'>*)
Combine duplicated requirements in a list.

Changed in version 0.2.1: Added the `normalize_func` keyword-only argument.

Changed in version 0.3.1: Requirements are no longer combined if their markers differ.

Parameters

- **requirement** (Union[str, Requirement, Iterable[Union[str, Requirement]]]) – A single requirement, or an iterable of requirements.
- **requirements** – Additional requirements.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of requirements. Default *shippinglabel.normalize()*.

Return type List[*ComparableRequirement*]

list_requirements(*name*, *depth=1*, *path=None*)

Returns an iterator over the requirements of the given library, and the requirements of those requirements.

The iterator is structured as follows:

```
[
    <requirement a>,
    [
        <requirement 1 of requirement a>,
        <requirement 2 of requirement a>,
        [<requirements of requirement 2>, ...],
        <requirement 3 of requirement a>,
    ],
    <requirement b>,
]
```

Parameters

- **name** (str)
- **depth** (int) – Default 1.
- **path** (Optional[Iterable[Union[str, Path, PathLike]]]) – The directories entries to search for distributions in. This can be used to search in a different (virtual) environment. Default sys.path.

Changed in version 0.8.2: The requirements are now sorted alphabetically.

Changed in version 1.7.0: Added the path argument.

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

parse_pyproject_dependencies(*pyproject_file*, *flavour='auto'*, ***, *normalize_func=<function 'normalize'>*)

Parse the project's dependencies from its pyproject.toml file.

New in version 0.10.0.

Parameters

- **pyproject_file** (Union[str, Path, PathLike])
- **flavour** (Literal['pep621', 'flit', 'auto']) – Either 'pep621' to parse from the **PEP 621** dependencies table, or 'flit' to parse the requires key in tool.flit.metadata`. The string ``'auto will use 'pep621' if available, otherwise try 'flit'. Default 'auto'.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of dependencies. Default *shippinglabel.normalize()*.

If no dependencies are defined an empty set is returned.

Return type Set[*ComparableRequirement*]

parse_pyproject_extras(*pyproject_file*, *flavour='auto'*, ***, *normalize_func=<function 'normalize'>*)
Parse the project's extra dependencies from its `pyproject.toml` file.

New in version 0.10.0.

Parameters

- **pyproject_file** (Union[str, Path, PathLike])
- **flavour** (Literal['pep621', 'flit', 'auto']) – Either `'pep621'` to parse from the **PEP 621** `dependencies` table, or `'flit'` to parse the `requires-extra` key in `tool.flit.metadata`. The string ``'auto` will use `'pep621'` if available, otherwise try `'flit'`. Default `'auto'`.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of dependencies. Default *shippinglabel.normalize()*.

If no extra dependencies are defined an empty dictionary is returned.

Return type Dict[str, Set[*ComparableRequirement*]]

parse_requirements(*requirements*, ***, *include_invalid=False*, *normalize_func=<function 'normalize'>*)
Parse the given strings as **PEP 508** requirements.

New in version 0.10.0.

Parameters

- **requirements** (Iterable[str])
- **include_invalid** (bool) – If `True`, include invalid lines as the third element of the tuple. Default `False`.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of requirements. Default *shippinglabel.normalize()*.

Return type Union[Tuple[Set[*ComparableRequirement*], List[str], List[str]], Tuple[Set[*ComparableRequirement*], List[str]]]

Returns The requirements, and a list of commented lines.

Overloads

- *parse_requirements*(**requirements**, **include_invalid**: Literal[True], **normalize_func** = ...) -> Tuple[Set[*ComparableRequirement*], List[str], List[str]]
- *parse_requirements*(**requirements**, **include_invalid**: Literal[False] = ..., **normalize_func** = ...) -> Tuple[Set[*ComparableRequirement*], List[str]]

read_requirements(*req_file*, *include_invalid=False*, *, *normalize_func=<function 'normalize'>*)

Reads **PEP 508** requirements from the given file.

Changed in version 0.2.0: Added the `include_invalid` option.

Changed in version 0.2.1: Added the `normalize_func` keyword-only argument.

Parameters

- **req_file** (Union[str, Path, PathLike])
- **include_invalid** (bool) – If True, include invalid lines as the third element of the tuple. Default False.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of requirements. Default *shippinglabel.normalize()*.

Return type Union[Tuple[Set[*ComparableRequirement*], List[str], List[str]], Tuple[Set[*ComparableRequirement*], List[str]]]

Returns The requirements, and a list of commented lines.

Overloads

- *read_requirements*(**req_file**, **include_invalid**: Literal[True], **normalize_func** = ...) -> Tuple[Set[*ComparableRequirement*], List[str], List[str]]
- *read_requirements*(**req_file**, **include_invalid**: Literal[False] = ..., **normalize_func** = ...) -> Tuple[Set[*ComparableRequirement*], List[str]]

resolve_specifiers(*specifiers*)

Resolve duplicated and overlapping requirement specifiers.

Parameters **specifiers** (Iterable[Specifier])

Return type SpecifierSet

Chapter

EIGHT

shippinglabel.sdist

Utilities for working with source distributions.

New in version 0.9.0.

NotAnSdistError(filename[, msg])	Raised when something other than a source distribution is passed to *parse_sdist_filename()*.
ParsedSdistFilename(project, version, extension)	Represents a parsed sdist filename.
parse_sdist_filename(filename)	Parse a sdist filename into its components.

exception NotAnSdistError (*filename*, *msg=''*)

Bases: ValueError

Raised when something other than a source distribution is passed to *parse_sdist_filename()*.

filename

Type: str

The invalid filename

namedtuple ParsedSdistFilename (*project*, *version*, *extension*)

Bases: NamedTuple

Represents a parsed sdist filename.

Fields

0) **project** (str) – The name of the project.
1) **version** (str) – The version number of the project.
2) **extension** (str) – The file extension of the sdist, e.g. .tar.gz.

parse_sdist_filename (*filename*)

Parse a sdist filename into its components.

Parameters **filename** (Union[str, Path, PathLike]) – An sdist path or filename.

Raises

- packaging.utils.InvalidSdistFilename if the filename is invalid.

  Changed in version 1.0.0: Previously raised wheel_filename.InvalidFilenameError

- *shippinglabel.sdist.NotAnSdistError* if the file is not an sdist.

Return type *ParsedSdistFilename*

See also:

packaging.utils.parse_sdist_filename()

Chapter

NINE

Extensions

These extensions add additional functionality but must be installed separately from PyPI.

9.1 shippinglabel_conda

Shippinglabel extension with utilities conda packages.

This extension must be installed separately:

9.1.1 from PyPI

```
$ python3 -m pip install shippinglabel-conda --user
```

9.1.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install shippinglabel-conda
```

__version__ = '0.1.0.post1'
The version number of this extension.

Data:

CONDA_API	Instance of apeye.slumber_url.SlumberURL for accessing the Conda API.

Functions:

clear_cache(*channel_name)	Clear the cached Conda channel listings.
compile_requirements(repo_dir[, extras])	Compile a list of requirements for the package from the `requirements.txt` file, and any extra dependencies.
get_channel_listing(channel_name)	Obtain the list of packages in the given Conda channel, either from the cache or from the Conda API.
make_conda_description(summary[, conda_channels])	Create a description for the Conda package from its summary and a list of channels required to install it.
prepare_requirements(requirements)	Prepare a list of requirements for use with conda.

continues on next page

Table 2 – continued from previous page

validate_requirements(requirements, ...)	Ensure that all requirements are available from the given Conda channels, and normalize the names to those in the Conda channel.

CONDA_API = SlumberURL('https://conda.anaconda.org')
Type: SlumberURL

Instance of apeye.slumber_url.SlumberURL for accessing the Conda API.

clear_cache(**channel_name*)
Clear the cached Conda channel listings.

Parameters ***channel_name** (str) – The name(s) of the channels to clear the cache for.

If no arguments are given the cache is cleared for all channels.

compile_requirements(*repo_dir*, *extras=()*)
Compile a list of requirements for the package from the `requirements.txt` file, and any extra dependencies.

Parameters

- **repo_dir** (PathPlus)
- **extras** (Iterable[str]) – A list of additional, optional requirements. These would be specified in "extras_require" for setuptools. Default `()`.

Return type List[*ComparableRequirement*]

get_channel_listing(*channel_name*)
Obtain the list of packages in the given Conda channel, either from the cache or from the Conda API.

Responses are cached for 48 hours. The cache can be cleared manually with *clear_cache()*.

Parameters **channel_name** (str)

Raises **ValueError** – if the channel can't be found.

Return type List[str]

make_conda_description(*summary*, *conda_channels=()*)
Create a description for the Conda package from its summary and a list of channels required to install it.

The description will look like:

```
This is my fancy Conda package. Hope you like it ☺.

Before installing please ensure you have added the following channels: conda-
↪forge, bioconda
```

if called as follows:

```
make_conda_description(
    "This is my fancy Conda package. Hope you like it ☺.",
    ["conda-forge", "bioconda"],
)
```

New in version 0.8.0.

Parameters

- **summary** (str)
- **conda_channels** (Iterable[str]) – Default ().

Return type str

prepare_requirements(*requirements*)
Prepare a list of requirements for use with conda.

This entails removing any extras and markers from the requirements, and skipping any requirements with URLs, as conda does not support these.

Parameters **requirements** (Iterable[*ComparableRequirement*])

Return type Iterable[*ComparableRequirement*]

validate_requirements(*requirements*, *conda_channels*)
Ensure that all requirements are available from the given Conda channels, and normalize the names to those in the Conda channel.

Parameters

- **requirements** (Iterable[*ComparableRequirement*])
- **conda_channels** (Iterable[str])

Return type List[*ComparableRequirement*]

9.2 shippinglabel_pypi

Shippinglabel extension for interacting with the Python Package Index (PyPI)..

See also:

pypi-json, which provides some of the functionality from this module but with a reusable HTTP session and support for authentication with other endpoints (such as a private package repository).

This extension must be installed separately:

9.2.1 from PyPI

```
$ python3 -m pip install shippinglabel-pypi --user
```

9.2.2 from Anaconda

First add the required channels

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

Then install

```
$ conda install shippinglabel-pypi
```

__version__ = '0.1.0.post1'
The version number of this extension.

Functions:

bind_requirements(filename[, specifier, ...])	Bind unbound requirements in the given file to the latest version on PyPI, and any later versions.
get_file_from_pypi(url, tmpdir)	Download the file with the given URL into the given (temporary) directory.
get_latest(pypi_name)	Returns the version number of the latest release on PyPI for the given project.
get_metadata(pypi_name)	Returns metadata for the given project on PyPI.
get_project_links(project_name)	Returns the web links for the given project.
get_pypi_releases(pypi_name)	Returns a dictionary mapping PyPI release versions to download URLs.
get_releases_with_digests(pypi_name)	Returns a dictionary mapping PyPI release versions to download URLs and the sha256sum of the file contents.
get_sdist_url(name, version[, strict])	Returns the URL of the project's source distribution on PyPI.
get_wheel_tag_mapping(name, version)	Constructs a mapping of wheel tags to the PyPI download URL of the wheel with relevant tag.
get_wheel_url(name, version[, strict])	Returns the URL of one of the project's wheels on PyPI.

bind_requirements (*filename*, *specifier='>='*, *normalize_func=<function 'normalize'>*)
Bind unbound requirements in the given file to the latest version on PyPI, and any later versions.

Parameters

- **filename** (Union[str, Path, PathLike]) – The requirements.txt file to bind requirements in.
- **specifier** (str) – The requirement specifier symbol to use. Default '>='.
- **normalize_func** (Callable[[str], str]) – Function to use to normalize the names of requirements. Default *shippinglabel.normalize()*.

Return type int

Returns 1 if the file was changed; 0 otherwise.

get_file_from_pypi (*url*, *tmpdir*)
Download the file with the given URL into the given (temporary) directory.

Parameters

- **url** (Union[URL, str]) – The URL to download the file from.

- **tmpdir** (Union[str, Path, PathLike]) – The (temporary) directory to store the downloaded file in.

get_latest(*pypi_name*)
Returns the version number of the latest release on PyPI for the given project.

Parameters **pypi_name** (str)

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- requests.HTTPError if an error occurs when communicating with PyPI.

Return type str

get_metadata(*pypi_name*)
Returns metadata for the given project on PyPI.

Parameters **pypi_name** (str)

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- requests.HTTPError if an error occurs when communicating with PyPI.

Return type Dict[str, Any]

get_project_links(*project_name*)
Returns the web links for the given project.

The exact keys vary, but common keys include "Documentation" and "Issue Tracker".

Note: The Home-Page field from Python core metadata is included under the Homepage key, if present. This matches the output parsed from PyPI for packages which are not installed.

Parameters **project_name** (str)

Return type MetadataMapping

get_pypi_releases(*pypi_name*)
Returns a dictionary mapping PyPI release versions to download URLs.

Parameters **pypi_name** (str) – The name of the project on PyPI.

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- requests.HTTPError if an error occurs when communicating with PyPI.

Return type Dict[str, List[str]]

get_releases_with_digests(*pypi_name*)
Returns a dictionary mapping PyPI release versions to download URLs and the sha256sum of the file contents.

Parameters **pypi_name** (str) – The name of the project on PyPI.

Raises

- packaging.requirements.InvalidRequirement if the project cannot be found on PyPI.
- requests.HTTPError if an error occurs when communicating with PyPI.

Return type Dict[str, List[FileURL]]

get_sdist_url(*name*, *version*, *strict=False*)
Returns the URL of the project's source distribution on PyPI.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])
- **strict** (bool) – Causes a ValueError to be raised if no sdist is found, rather than retuning a wheel. Default False.

> **Attention:** If no source distribution is found this function may return a wheel or "zip" sdist unless strict is True.

Return type str

get_wheel_tag_mapping(*name*, *version*)
Constructs a mapping of wheel tags to the PyPI download URL of the wheel with relevant tag.

This can be used alongside packaging.tags.sys_tags() to select the best wheel for the current platform.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])

Return type Tuple[Dict[Tag, URL], List[URL]]

Returns A tuple containing the tag: url mapping, and a list of download URLs for non-wheel artifacts (e.g. sdists).

get_wheel_url(*name*, *version*, *strict=False*)
Returns the URL of one of the project's wheels on PyPI.

For finer control over which wheel the URL is for see the *get_wheel_tag_mapping()* function.

Parameters

- **name** (str) – The name of the project on PyPI.
- **version** (Union[str, int, Version])
- **strict** (bool) – Causes a ValueError to be raised if no wheels are found, rather than retuning a wheel. Default False.

Attention: If no wheels are found this function may return an sdist unless `strict` is `True`.

Return type `str`

Python Module Index

S

Index

Symbols

B

C

E

F

G

W