wheelfile

MrMino

Aug 06, 2021

CONTENTS

API for handling ".whl" files.

Use *WheelFile* to create or read a wheel.

Managing metadata is done via *metadata*, *wheeldata*, and *record* attributes. See *MetaData*, *WheelData*, and *WheelRecord* for documentation of the objects returned by these attributes.

Example

Here's how to create a simple package under a specific directory path:

```
with WheelFile('path/to/directory/', mode='w'
               distname="mywheel", version="1") as wf:
    wf.write('path/to/a/module.py', arcname="mywheel.py")
```

CHAPTER

ONE

INSTALLATION

To be able to use the module, you have to install it first:

```
pip install wheelfile
```

CHAPTER

TWO

MAIN CLASS

class wheelfile.**WheelFile**(*file_or_path: Union[str, pathlib.Path, BinaryIO] = './'*, *mode: str = 'r'*, ***, *distname: Optional[str] = None*, *version: Optional[Union[str, packaging.version.Version]] = None*, *build_tag: Optional[Union[int, str]] = None*, *language_tag: Optional[str] = None*, *abi_tag: Optional[str] = None*, *platform_tag: Optional[str] = None*)

An archive that follows the wheel specification.

Used to read, create, validate, or modify *.whl* files.

Can be used as a context manager, in which case *close()* is called upon exiting the context.

filename

Path to the file, if the instance was initialized with one, otherwise None.

Type str

distname

Name of the distribution (project). Either given to __init__() explicitly or inferred from its file_or_path argument.

Type str

version

Version of the distribution. Either given to __init__() explicitly or inferred from its file_or_path argument.

Type packaging.version.Version

build_tag

Distribution's build number. Either given to __init__() explicitly or inferred from its file_or_path argument, otherwise *None* in lazy mode.

Type Optional[int]

language_tag

Interpretter implementation compatibility specifier. See PEP-425 for the full specification. Either given to __init__() explicitly or inferred from its file_or_path argument otherwise an empty string in lazy mode.

Type str

abi_tag

ABI compatibility specifier. See PEP-425 for the full specification. Either given to __init__() explicitly or inferred from its file_or_path argument, otherwise an empty string in lazy mode.

Type str

platform_tag

Platform compatibility specifier. See PEP-425 for the full specification. Either given to __init__() explicitly or inferred from its file_or_path argument, otherwise an empty string in lazy mode.

Type str

record
Current state of .dist-info/RECORD file.

When reading wheels in lazy mode, if the file does not exist or is misformatted, this attribute becomes None.

In non-lazy modes this file is always read & validated on initialization. In write and exclusive-write modes, written to the archive on close().

Type Optional[*WheelRecord*]

metadata
Current state of .dist-info/METADATA file.

Values from *distname* and *version* are used to provide required arguments when the file is created from scratch by *__init__()*.

When reading wheels in lazy mode, if the file does not exist or is misformatted, this attribute becomes None.

In non-lazy modes this file is always read & validated on initialization. In write and exclusive-write modes, written to the archive on close().

Type Optional[*MetaData*]

wheeldata
Current state of .dist-info/WHEELDATA file.

Values from *build_tag*, *language_tag*, *abi_tag*, *platform_tag*, or their substitutes inferred from the filename are used to initialize this object.

When reading wheels in lazy mode, if the file does not exist or is misformatted, this attribute becomes None.

In non-lazy modes this file is always read & validated on initialization. In write and exclusive-write modes, written to the archive on close().

Type Optional[*WheelData*]

closed
True if the underlying *ZipFile* object is closed, false otherwise.

Type bool

__init__(*file_or_path: Union[str, pathlib.Path, BinaryIO] = './'*, *mode: str = 'r'*, ***, *distname: Optional[str] = None*, *version: Optional[Union[str, packaging.version.Version]] = None*, *build_tag: Optional[Union[int, str]] = None*, *language_tag: Optional[str] = None*, *abi_tag: Optional[str] = None*, *platform_tag: Optional[str] = None*) → None
Open or create a wheel file.

In write and exclusive-write modes, if *file_or_path* is not specified, or the specified path is a directory, the wheelfile will be created in the current working directory, with filename generated using the values given via *distname*, *version*, *build_tag*, *language_tag*, *abi_tag*, and *platfrom_tag* arguments. Each of these parameters is stored in a read-only property of the same name.

If lazy mode is not specified:

- In read and append modes, the file is validated using validate(). Contents of metadata files inside .dist-info directory are read and converted into their respective object representations (see "metadata", "wheeldata", and "record" attributes).
- In write and exclusive-write modes, object representations for each metadata file are created from scratch. They will be written to each of their respective .dist-info/ files on close().

To skip the validation, e.g. if you wish to fix a misformated wheel, use lazy mode ('l' - see description of the "mode" parameter).

In lazy mode, if the opened file does not contain WHEEL, METADATA, or RECORD (which is optional as per PEP-627), the attributes corresponding to the missing data structures will be set to None.

If any of the metadata files cannot be read due to a wrong format, they are considered missing.

Filename tags are only inferred if the filename contains 5 or 6 segments inbetween '-' characters. Otherwise, if any tag argument is omitted, its attribute is set to an empty string.

If the archive root contains a directory with a name ending with '.dist-info', it is considered to be _the_ metadata directory for the wheel, even if the given/inferred distname and version do not match its name.

If the archive already contains either one of the aforementioned files, they are read, but are not checked for consistency. Use validate() to check whether there are errors, and fix() to fix them.

There are currently 2 classes of errors which completely prevent a well formatted zip file from being read by this class:

- Unknown/incorrect distribution name/version - when the naming scheme is violated in a way that prevents inferring these values and the user hasn't provided these values, or provided ones that do not conform to the specifications. In such case, the scope of functioning features of this class would be limited to that of a standard ZipFile, and is therefore unsupported.
- When there are multiple .data or .dist-info directories. This would mean that the class would have to guess which are the genuine ones - and we refuse the temptation to do that (see "The Zen of Python").

In other words, this class is liberal in what it accepts, but very conservative in what it does (A.K.A. the robustness principle).

Note: Despite all of this, THERE ARE NO GUARANTEES being made as to whether a misformatted file can be read or fixed by this class, and even if it is currently, whether it will still be the case in the future versions.

Parameters

- **file_or_path** – Path to the file to open/create or a file-like object to use.
- **mode** – See zipfile.ZipFile docs for the list of available modes.

  In the read and append modes, the file given has to contain proper PKZIP-formatted data.

  Adding "l" to the mode string turns on the "lazy mode". This changes the behavior on initialization (see above), the behavior of close() (see its docstring for more info), makes the archive modifying methods refrain from refreshing the record & writing it to the archive.

  Lazy mode should only be used in cases where a misformatted wheels have to be read or fixed.

- **distname** – Name of the distribution for this wheelfile.

  If omitted, the name will be inferred from the filename given in the path. If a file-like object is given instead of a path, it will be inferred from its "name" attribute.

  The class requires this information, as it's used to infer the name of the directory in the archive in which metadata should reside.

  This argument should be understood as an override for the values calculated from the object given in "file_or_path" argument. It should only be necessary when a file is read from memory or has a misformatted name.

Should be composed of alphanumeric characters and underscores only. Must not be an empty string.

See the description of "distname" attribute for more information.

- **version** – Version of the distribution in this wheelfile. Follows the same semantics as "distname".

  The given value must be compliant with PEP-440 version identifier specification.

  See the description of "version" attribute for more information.

- **build** – Optional build number specifier for the distribution.

  See *WheelData* docstring for information about semantics of this field.

  If lazy mode is not specified, this value must be an integer or a string that converts to one. Otherwise no checks for this value are performed.

- **language_tag** – Language implementation specification. Used to distinguish between distributions targetted at different versions of interpreters.

  The given value should be in the same form as the ones appearing in wheels' filenames.

  Defaults to *'py3'*, but only if an unnamed or a directory target was given.

- **abi_tag** – In distributions that utilize compiled binaries, specifies the version of the ABI that the binaries in the wheel are compatible with.

  The given value should be in the same form as the ones appearing in wheels' filenames.

  Defaults to *'none'*, but only if an unnamed or a directory target was given.

- **platform_tag** – Used to specify platforms that the distribution is compatible with.

  The given value should be in the same form as the ones appearing in wheels' filenames.

  Defaults to *'any'*, but only if an unnamed or a directory target was given.

Raises

- *UnnamedDistributionError* – Raised if the distname or version cannot be inferred from the given arguments.

  E.g. when the path does not contain the version, or the file-like object has no "name" attribute to get the filename from, and the information wasn't provided via other arguments.

- *BadWheelFileError* – Raised if the archive contains multiple '.dist-info' or '.data' directories.

- **zipfile.BadZipFile** – If given file is not a proper zip.

__weakref__
list of weak references to the object (if defined)

close() → None
Finalize and close the file.

Writes the rest of the necessary data to the archive, performs one last validation of the contents (unless the file is open in lazy mode), and closes the file.

There must not be any handles left open for the zip contents, i.e. all objects returned by *open()* or *.zip.open()* must be closed before calling this subroutine.

Raises **RuntimeError** – If there are unclosed content handles.

write(*filename: Union[str, pathlib.Path]*, *arcname: Optional[str] = None*, ***, *recursive: bool = True*, *resolve: bool = True*) → None

Add the file to the wheel.

Updates the wheel record, if the record is being kept.

Parameters

- **filename** – Path to the file or directory to add.
- **arcname** – Path in the archive to assign the file/directory into. If not given, *filename* will be used instead. In both cases, the leading path separators and the drive letter (if any) will be removed.
- **recursive** – Keyword only. When True, if given path leads to a directory, all of its contents are going to be added into the archive, including contents of its subdirectories.

  If its False, only a directory entry is going to be added, without any of its contents.
- **resolve** – Keyword only. When True, and no *arcname* is given, the path given to *filename* will not be used as the arcname (as is the case with *ZipFile.write*), but only the name of the file that it points to will be used.

  For example, if you set *filename* to *../some/other/dir/file*, *file* entry will be written in the archive root.

  Has no effect when set to False or when *arcname* is given.

> **Warning:** This prevents adding an unresolved path to the wheel by accident. Setting this to False might lead to ./ or ../ being added to the wheel, which will lead to loss of data when users try to update/remove the distribution from their environment.

write_data(*filename: Union[str, pathlib.Path]*, *section: str*, *arcname: Optional[str] = None*, ***, *recursive: bool = True*, *resolve: bool = True*) → None

Write a file to the .data directory under a specified section.

This method is a handy shortcut for writing into *<dist>-<version>.data/*, such that you dont have to generate the path yourself.

Updates the wheel record, if the record is being kept.

Parameters

- **filename** – Path to the file or directory to add.
- **section** – Name of the section, i.e. the directory inside *.data/* that the file should be put into. Sections have special meaning, see PEP-427. Cannot contain any slashes, nor be empty.
- **arcname** – Path in the archive to assign the file/directory into, relative to the directory of the specified data section. If left empty, filename is used. Leading slashes are stripped.
- **recursive** – Keyword only. When True, if given path leads to a directory, all of its contents are going to be added into the archive, including contents of its subdirectories.

  If its False, only a directory entry is going to be added, without any of tis contents.
- **resolve** – Keyword only. When True, and no *arcname* is given, the path given to *filename* will not be used as the arcname (as is the case with *ZipFile.write*), but only the name of the file that it points to will be used.

For example, if you set *filename* to *../some/other/dir/file*, *file* entry will be written in the archive root.

Has no effect when set to False or when *arcname* is given.

> **Warning:** This prevents adding an unresolved path to the wheel by accident. Setting this to False might lead to ./ or ../ being added to the wheel, which will lead to loss of data when users try to update/remove the distribution from their environment.

write_distinfo(*filename: Union[str, pathlib.Path]*, *arcname: Optional[str] = None*, *, *recursive: bool = True*, *resolve: bool = True*) → None

Write a file to *.dist-info* directory in the wheel.

This is a shorthand for *write(...)* with *arcname* prefixed with the *.dist-info* path. It also ensures that the metadata files critical to the wheel correctnes (i.e. the ones written into archive on *close()*) aren't being pre-writen.

Parameters

- **filename** – Path to the file or directory to add.
- **arcname** – Path in the archive to assign the file/directory into. If not given, *filename* will be used instead. In both cases, the leading path separators and the drive letter (if any) will be removed.

  This parameter will be prefixed with proper *.dist-info* path automatically.
- **recursive** – Keyword only. When True, if given path leads to a directory, all of its contents are going to be added into the archive, including contents of its subdirectories.

  If its False, only a directory entry is going to be added, without any of its contents.
- **resolve** – Keyword only. When True, and no *arcname* is given, the path given to *filename* will not be used as the arcname (as is the case with *ZipFile.write*), but only the name of the file that it points to will be used.

  For example, if you set *filename* to *../some/other/dir/file*, *file* entry will be written in the *.dist-info* directory.

  Has no effect when set to False or when *arcname* is given.

Raises **ProhibitedWriteError** – Raised if the write would result with duplicated *WHEEL*, *METADATA*, or *RECORD* files after *close()* is called.

writestr(*zinfo_or_arcname: Union[zipfile38.ZipInfo, str]*, *data: Union[bytes, str]*) → None

Write given data into the wheel under the given path.

Updates the wheel record, if the record is being kept.

Parameters

- **zinfo_or_arcname** – Specifies the path in the archive under which the data will be stored.
- **data** – The data that will be writen into the archive. If it's a string, it is encoded as UTF-8 first.

writestr_data(*section: str*, *zinfo_or_arcname: Union[zipfile38.ZipInfo, str]*, *data: Union[bytes, str]*) → None

Write given data to the .data directory under a specified section.

This method is a handy shortcut for writing into *<dist>-<version>.data/*, such that you dont have to generate the path yourself.

Updates the wheel record, if the record is being kept.

Parameters

- **section** – Name of the section, i.e. the directory inside *.data/* that the file should be put into. Sections have special meaning, see PEP-427. Cannot contain any slashes, nor be empty.
- **zinfo_or_arcname** – Specifies the path in the archive under which the data will be stored. This is relative to the path of the section directory. Leading slashes are stripped.
- **data** – The data that will be writen into the archive. If it's a string, it is encoded as UTF-8 first.

CHAPTER

THREE

METADATA CLASSES

class wheelfile.**WheelRecord**(*hash_algo: str = 'sha256'*)

Contains logic for creation and modification of RECORD files.

Keeps track of files in the wheel and their hashes.

For the full spec, see PEP-376 "RECORD" section, PEP-627, "The .dist-info directory" section of PEP-427, and https://packaging.python.org/specifications/recording-installed-packages/.

__eq__(*other*)

Return self==value.

__init__(*hash_algo: str = 'sha256'*)

__str__() → str

Return str(self).

__weakref__

list of weak references to the object (if defined)

property hash_algo: str

Hash algorithm to use to generate RECORD file entries

hash_of(*arcpath*) → str

Return the hash of a file in the archive this RECORD describes

Parameters **arcpath** – Location of the file inside the archive.

Returns String in the form <algorithm>=<base64_str>, where algorithm is the name of the hashing agorithm used to generate the hash (see hash_algo), and base64_str is a string containing a base64 encoded version of the hash with any trailing '=' removed.

Return type str

update(*arcpath: str*, *buf: IO[bytes]*)

Add a record entry for a file in the archive.

Parameters **buf** – Buffer from which the data will be read in HASH_BUF_SIZE chunks. Must be fresh, i.e. seek(0)-ed.

class wheelfile.**WheelData**(**, generator: str = 'wheelfile 0.0.6', root_is_purelib: bool = True, tags: Union[List[str], str] = 'py3-none-any', build: Optional[int] = None*)

Implements .dist-info/WHEEL file format.

Descriptions of parameters based on PEP-427. All parameters are keyword only. Attributes of objects of this class follow parameter names.

Note: Wheel-Version, the wheel format version specifier, is unchangeable. Version "1.0" is used.

Parameters

- **generator** – Name and (optionally) version of the generator that generated the wheel file. By default, "wheelfile {__version__}" is used.
- **root_is_purelib** – Defines whether the root of the wheel file should be first unpacked into purelib directory (see distutils.command.install.INSTALL_SCHEMES).
- **tags** – See PEP-425 - "Compatibility Tags for Built Distributions". Either a single string denoting one tag or a list of tags. Tags may contain compressed tag sets, in which case they will be expanded.

  By default, "py3-none-any" is used.
- **build** – Optional build number. Used as a tie breaker when two wheels have the same version.

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

`__init__`(*, *generator: str = 'wheelfile 0.0.6'*, *root_is_purelib: bool = True*, *tags: Union[List[str], str] = 'py3-none-any'*, *build: Optional[int] = None*)

`__str__`() → str
Return str(self).

class wheelfile.**MetaData**(*, *name: str*, *version: Union[str, packaging.version.Version]*, *summary: Optional[str] = None*, *description: Optional[str] = None*, *description_content_type: Optional[str] = None*, *keywords: Optional[Union[List[str], str]] = None*, *classifiers: Optional[List[str]] = None*, *author: Optional[str] = None*, *author_email: Optional[str] = None*, *maintainer: Optional[str] = None*, *maintainer_email: Optional[str] = None*, *license: Optional[str] = None*, *home_page: Optional[str] = None*, *download_url: Optional[str] = None*, *project_urls: Optional[List[str]] = None*, *platforms: Optional[List[str]] = None*, *supported_platforms: Optional[List[str]] = None*, *requires_python: Optional[str] = None*, *requires_dists: Optional[List[str]] = None*, *requires_externals: Optional[List[str]] = None*, *provides_extras: Optional[List[str]] = None*, *provides_dists: Optional[List[str]] = None*, *obsoletes_dists: Optional[List[str]] = None*)

Implements Wheel Metadata format v2.1.

Descriptions of parameters based on https://packaging.python.org/specifications/core-metadata/. All parameters are keyword only. Attributes of objects of this class follow parameter names.

All parameters except "name" and "version" are optional.

Note: Metadata-Version, the metadata format version specifier, is unchangable. Version "2.1" is used.

Parameters

- **name** – Primary identifier for the distribution that uses this metadata. Must start and end with a letter or number, and consists only of ASCII alphanumerics, hyphen, underscore, and period.
- **version** – A string that contains PEP-440 compatible version identifier.

  Can be specified using packaging.version.Version object, or a string, where the latter is always converted to the former.
- **summary** – A one-line sentence describing this distribution.

- **description** – Longer text that describes this distribution in detail. Can be written using plaintext, reStructuredText, or Markdown (see "description_content_type" parameter below).

  The string given for this field should not include RFC 822 indentation followed by a "|" symbol. Newline characters are permitted

- **description_content_type** – Defines content format of the text put in the "description" argument. The field value should follow the following structure:

  <type/subtype>; charset=<charset>[; <param_name>=<param value> ...]

  Valid type/subtype strings are:

  - text/plain
  - text/x-rst
  - text/markdown

  For charset parameter, the only legal value is UTF-8.

  For text/markdown, parameter "variant=<variant>" specifies variant of the markdown used. Currently recognized variants include "GFM" and "CommonMark".

  Examples:

  Description-Content-Type: text/markdown; charset=UTF-8; variant=GFM

  Description-Content-Type: text/markdown

- **keywords** – List of search keywords for this distribution. Optionally a single string literal with keywords separated by commas.

  Note: despite the name being a plural noun, the specification defines this field as a single-use field. In this implementation however, the value of the attribute after instance initialization is a list of strings, and conversions to and from string follow the spec - they require a comma-separated list.

- **classifiers** – List PEP-301 classification values for this distribution, optionally followed by a semicolon and an environmental marker.

  Example of a classifier:

  Operating System :: Microsoft :: Windows :: Windows 10

- **author** – Name and, optionally, contact information of the original author of the distribution.

- **author_email** – Email address of the person specified in the "author" parameter. Format of this field must follow the format of RFC-822 "From:" header field.

- **maintainer** – Name and, optionally, contact information of person currently maintaining the project to which this distribution belongs to.

  Omit this parameter if the author and current maintainer is the same person.

- **maintainer_email** – Email address of the person specified in the "maintainer" parameter. Format of this field must follow the format of RFC-822 "From:" header field.

  Omit this parameter if the author and current maintainer is the same person.

- **license** – Text of the license that covers this distribution. If license classifier is used, this parameter may be omitted or used to specify the particular version of the intended legal text.

- **home_page** – URL of the home page for this distribution (project).

- **download_url** – URL from which this distribution (in this version) can be downloaded.
- **project_urls** – List of URLs with labels for them, in the following format:

  <label>, <url>

  The label must be at most 32 characters.

  Example of an item of this list:

  Repository, https://github.com/MrMino/wheelfile
- **platforms** – List of strings that signify supported operating systems. Use only if an OS cannot be listed by using a classifier.
- **supported_platforms** – In binary distributions list of strings, each defining an operating system and a CPU for which the distribution was compiled.

  Semantics of this field aren't formalized by metadata specifications.
- **requires_python** – PEP-440 version identifier, that specifies the set Python language versions that this distribution is compatible with.

  Some package management tools (most notably pip) use the value of this field to filter out installation candidates.

  Example:

  ~=3.5,!=3.5.1,!=3.5.0
- **requires_dists** – List of PEP-508 dependency specifiers (think line-split contents of requirements.txt).
- **requires_externals** – List of system dependencies that this distribution requires.

  Each item is a string with a name of the dependency optionally followed by a version (in the same way items in "requires_dists") are specified.

  Each item may end with a semicolon followed by a PEP-496 environment markers.
- **provides_extras** – List of names of optional features provided by a distribution. Used to specify which dependencies should be installed depending on which of these optional features are requested.

  For example, if you specified "network" and "ssh" as optional features, the following requirement specifier can be used in "requires_externals" list to indicate, that the "paramiko" dependency should only be installed when "ssh" feature is requested:

  paramiko; extra == "ssh"

  or

  paramiko[ssh]

  If a dependency is required by multiple features, the features can be specified in a square brackets, separated by commas:

  ipython[repl, jupyter_kernel]

  Specifying an optional feature without using it in "requires_externals" is considered invalid.

  Feature names "tests" and "doc" are reserved in their semantics. They can be used for dependencies of automated testing or documentation generation.
- **provides_dists** – List of names of other distributions contained within this one. Each entry must follow the same format that entries in "requires_dists" list do.

Different distributions may use a name that does not correspond to any particular project, to indicate a capability to provide a certain feature, e.g. "relational_db" may be used to say that a project provides relational database capabilities

- **obsoletes_dists** – List of names of distributions obsoleted by installing this one, indicating that they should not coexist in a single environment with this one. Each entry must follow the same format that entries in "requires_dists" list do.

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

__init__(*, *name: str*, *version: Union[str, packaging.version.Version]*, *summary: Optional[str] = None*, *description: Optional[str] = None*, *description_content_type: Optional[str] = None*, *keywords: Optional[Union[List[str], str]] = None*, *classifiers: Optional[List[str]] = None*, *author: Optional[str] = None*, *author_email: Optional[str] = None*, *maintainer: Optional[str] = None*, *maintainer_email: Optional[str] = None*, *license: Optional[str] = None*, *home_page: Optional[str] = None*, *download_url: Optional[str] = None*, *project_urls: Optional[List[str]] = None*, *platforms: Optional[List[str]] = None*, *supported_platforms: Optional[List[str]] = None*, *requires_python: Optional[str] = None*, *requires_dists: Optional[List[str]] = None*, *requires_externals: Optional[List[str]] = None*, *provides_extras: Optional[List[str]] = None*, *provides_dists: Optional[List[str]] = None*, *obsoletes_dists: Optional[List[str]] = None*)

__str__() → str
Return str(self).

CHAPTER

FOUR

EXCEPTIONS

exception wheelfile.**BadWheelFileError**
The given file cannot be interpreted as a wheel nor fixed.

exception wheelfile.**UnnamedDistributionError**
Distribution name cannot be deduced from arguments.

exception wheelfile.**ProhibitedWriteError**
Writing into given arcname would result in a corrupted package.

CHAPTER

FIVE

UTILITIES

wheelfile.**resolved**(*path: Union[str, pathlib.Path]*) → str

Get the name of the file or directory the path points to.

This is a convenience function over the functionality provided by *resolve* argument of *WheelFile.write* and similar methods. Since *resolve=True* is ignored when *arcname* is given, it is impossible to add arbitrary prefix to *arcname* without resolving the path first - and this is what this function provides.

Using this, you can have both custom *arcname* prefix and the "resolve" functionality, like so:

```
wf = WheelFile(...)
wf.write(some_path, arcname="arc/dir/" + resolved(some_path))
```

Parameters **path** – Path to resolve.

Returns The name of the file or directory the *path* points to.

Return type str

PYTHON MODULE INDEX

INDEX

Symbols

A

B

C

D

F

H

L

M

P

R

U

V

W