

---

**wheelfile**

**MrMino**

**Aug 06, 2021**



# CONTENTS

<b>1</b>	<b>Installation</b>	<b>3</b>
<b>2</b>	<b>Main class</b>	<b>5</b>
<b>3</b>	<b>Metadata classes</b>	<b>15</b>
<b>4</b>	<b>Exceptions</b>	<b>21</b>
<b>5</b>	<b>Utilities</b>	<b>23</b>
	<b>Python Module Index</b>	<b>25</b>
	<b>Index</b>	<b>27</b>



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")
```



## INSTALLATION

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

```
pip install wheelfile
```





## 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,
    compression: int = 8, allowZip64: bool = True, compresslevel: Optional[int] =
    None, strict_timestamps: bool = True)
```

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****If initialized with:**

- An IO buffer: the filename that the wheel would have after saving it onto filesystem.
- A path to a directory: the path to the wheel, composed from its the given path and the filename generated using the parameters given to `__init__`.
- A path to a file: that path, even if it is not compliant with the spec (in lazy mode).

**Returned path is not resolved, and so might be relative and/or contain `../` and `./` segments.**

**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**

Interpreter 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\*](#)]

**distinfo\_dirname**

Name of the `.dist-info` directory inside the archive wheel, without the trailing slash.

**data\_dirname**

Name of the `.data` directory inside the archive wheel, without the trailing slash.

**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, compression: int = 8, allowZip64: bool = True, compresslevel: Optional[int] = None, strict_timestamps: bool = True) → None
```

Open or create a wheel file.

In write and exclusive-write modes, if *file\_or\_path* is not specified, it is assumed to be the current directory. If the specified path is a directory, the wheelfile will be created inside it, with filename generated using the values given via *distname*, *version*, *build\_tag*, *language\_tag*, *abi\_tag*, and *platform\_tag* arguments. Each of these parameters is stored in a read-only property of the same name. If *file\_or\_path* is a path to a file, the wheel will be created under the specified path.

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 misformatted 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.

- **compression** – Compression method to use. By default `zipfile.ZIP_DEFLATED` is used.

See *zipfile.ZipFile* documentation for the full description. This argument differs from its *ZipFile* counterpart in that here it is keyword-only, and the default value is different.

- **allowZip64** – Flag used to indicate whether ZIP64 extensions should be used.  
See *zipfile.ZipFile* documentation for the full description. This argument differs from its *ZipFile* counterpart in that here it is keyword-only.
- **compresslevel** – Compression level to use when writing to the archive.  
See *zipfile.ZipFile* documentation for the full description. This argument differs from its *ZipFile* counterpart in that here it is keyword-only
- **strict\_timestamps** – When *True*, files with modification times older than 1980-01-01 or newer than 2107-12-31 are allowed. Keyword only.  
See *zipfile.ZipFile* documentation for the full description. This argument differs from its *ZipFile* counterpart in that here it is keyword-only.

#### 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.

```
classmethod from_wheelfile(wf: wheelfile.WheelFile, file_or_path: Union[str, pathlib.Path, BinaryIO]
    = './', mode: str = 'w', *, distname: Union[str, None,
    wheelfile.WheelFile._Sentinel] = <wheelfile.WheelFile._Sentinel object>,
    version: Union[str, packaging.version.Version, None,
    wheelfile.WheelFile._Sentinel] = <wheelfile.WheelFile._Sentinel object>,
    build_tag: Union[int, str, None, wheelfile.WheelFile._Sentinel] =
    <wheelfile.WheelFile._Sentinel object>, language_tag: Union[str, None,
    wheelfile.WheelFile._Sentinel] = <wheelfile.WheelFile._Sentinel object>,
    abi_tag: Union[str, None, wheelfile.WheelFile._Sentinel] =
    <wheelfile.WheelFile._Sentinel object>, platform_tag: Union[str, None,
    wheelfile.WheelFile._Sentinel] = <wheelfile.WheelFile._Sentinel object>,
    compression: int = 8, allowZip64: bool = True, compresslevel:
    Optional[int] = None, strict_timestamps: bool = True) →
    wheelfile.WheelFile
```

Recreate *wf* using different parameters.

Creates a new *WheelFile* object using data from another, given as *wf*, and given constructor parameters. The new object will contain the same files (including those under *.data* and *.dist-info* directories), with the same timestamps, compression methods, compression levels, access modes, etc., *except*:

- *.dist-info* directory is renamed, if *version* or *distname* is changed.
- *.data* directory is renamed, if *version* or *distname* is changed.
- *METADATA* will contain almost all the same information, except for the fields that were changed via arguments of this method.
- *WHEEL* will be changed to contain the new *tags* and *build* number. The *generator* field will be reset to the one given by *WheelData* by default: “*wheelfile <version>*”.
- *RECORD* is reconstructed in order to accomodate the differences above.

This can be used to rename a wheel, change its metadata, or add files to it. It also fixes some common problems of wheel packages, e.g. unnormalized *dist-info* directory names.

If any of the metadata files is missing or corrupted in *wf*, i.e. the properties *metadata*, *wheeldata*, and *record* of *wf* are set to *None*, new ones will be created for the new object, using default values.

All parameters of this method except *wf* are passed to the new object’s `__init__`.

For *distname*, *version*, and *\*\_tag* arguments, if a parameter is not given, the value from *wf* is used. If a default is needed instead, set the argument to *None* explicitly.

Even if *wf* was created using a path to a directory, the default value used for *file\_or\_path* will be the current working directory.

The new *WheelFile* object must be writable, so by default “w” mode is used, instead of “r”. If copying *wf* would result in overwriting a file or buffer from which *wf* was created, *ValueError* will be raised.

#### Parameters

- **wf** – *WheelFile* object from which the new object should be recreated.
- **mode** – Mode in which the new *WheelFile* should be opened. Accepts the same modes as *WheelFile.\_\_init\_\_*, except read mode (“r”) is not allowed.
- **distname** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is available via property of the same name). If *None* is given explicitly, constructor’s default is used.
- **version** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is available via property of the same name). If *None* is given explicitly, constructor’s default is used.
- **build\_tag** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is available via property of the same name). If *None* is given explicitly, constructor’s default is used.
- **language\_tag** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is available via property of the same name). If *None* is given explicitly, constructor’s default is used.
- **abi\_tag** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is available via property of the same name). If *None* is given explicitly, constructor’s default is used.
- **platform\_tag** – Optional argument passed to the new object’s constructor. See *WheelFile.\_\_init\_\_* for the full description. If not specified, value from *wf* is used (it is

available via property of the same name). If *None* is given explicitly, constructor's default is used.

- **compression** – Optional argument passed to the new object's constructor, which in turn passes them to *zipfile.ZipFile* - see *zipfile* docs for full description on each.

Value from *wf* is *not* reused for this parameter.

- **allowZip64** – Optional argument passed to the new object's constructor, which in turn passes them to *zipfile.ZipFile* - see *zipfile* docs for full description on each.

Value from *wf* is *not* reused for this parameter.

- **compresslevel** – Optional argument passed to the new object's constructor, which in turn passes them to *zipfile.ZipFile* - see *zipfile* docs for full description on each.

Value from *wf* is *not* reused for this parameter.

- **strict\_timestamps** – Optional argument passed to the new object's constructor, which in turn passes them to *zipfile.ZipFile* - see *zipfile* docs for full description on each.

Value from *wf* is *not* reused for this parameter.

**Raises ValueError** – Raised when: - Read mode is used ("r") in *mode*. - Creating the object would result in rewriting the underlying buffer of *wf* (if *wf* uses an IO buffer). - Creating the object would result in rewriting the file on which *wf* operates.

**infolist()** → List[zipfile38.ZipInfo]

Return a list of ZipInfo objects for each wheel member.

Same as ZipFile.infolist(), but omits objects corresponding to RECORD, METADATA, and WHEEL files.

**namelist()** → List[str]

Return a list of wheel members by name, omit metadata files.

Same as ZipFile.namelist(), but omits RECORD, METADATA, and WHEEL files.

**write**(filename: Union[str, pathlib.Path], arcname: Optional[str] = None, compress\_type: Optional[int] = None, compresslevel: Optional[int] = None, \*, recursive: bool = True, resolve: bool = True, skipdir: 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.
- **compress\_type** – Same as in *zipfile.ZipFile.write*. Overrides the *compression* parameter given to *\_\_init\_\_*.
- **compresslevel** – Same as in *zipfile.ZipFile.write*. Overrides the *compresslevel* parameter given to *\_\_init\_\_*.
- **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.

- **skipdir** – Keyword only. Indicates whether directory entries should be skipped in the archive. Set to *True* by default, which means that attempting to write an empty directory will be silently omitted.

**write\_data**(*filename*: Union[str, pathlib.Path], *section*: str, *arcname*: Optional[str] = None, *compress\_type*: Optional[int] = None, *compresslevel*: Optional[int] = None, \*, *recursive*: bool = True, *resolve*: bool = True, *skipdir*: 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 don't 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.
- **compress\_type** – Same as in *zipfile.ZipFile.write*. Overrides the *compression* parameter given to *\_\_init\_\_*.
- **compresslevel** – Same as in *zipfile.ZipFile.write*. Overrides the *compresslevel* parameter given to *\_\_init\_\_*.
- **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.

- **skipdir** – Keyword only. Indicates whether directory entries should be skipped in the archive. Set to *True* by default, which means that attempting to write an empty directory will be silently omitted.

**write\_distinfo**(*filename*: Union[str, pathlib.Path], *arcname*: Optional[str] = None, *compress\_type*: Optional[int] = None, *compresslevel*: Optional[int] = None, \*, *recursive*: bool = True, *resolve*: bool = True, *skipdir*: 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 correctness (i.e. the ones written into archive on `close()`) aren't being pre-written.

### 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.

- **compress\_type** – Same as in `zipfile.ZipFile.write`. Overrides the `compression` parameter given to `__init__`.
- **compresslevel** – Same as in `zipfile.ZipFile.write`. Overrides the `compresslevel` parameter given to `__init__`.
- **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.

- **skipdir** – Keyword only. Indicates whether directory entries should be skipped in the archive. Set to `True` by default, which means that attempting to write an empty directory will be silently omitted.

**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]`, `compress_type: Optional[int] = None`, `compresslevel: Optional[int] = None`) → 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 written into the archive. If it's a string, it is encoded as UTF-8 first.
- **compress\_type** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compression` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compress_type` field is also overridden.
- **compresslevel** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compresslevel` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compresslevel` field is also overridden.

**writestr\_data**(*section: str, zinfo\_or\_arcname: Union[zipfile38.ZipInfo, str], data: Union[bytes, str], compress\_type: Optional[int] = None, compresslevel: Optional[int] = None*) → 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 don't 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 written into the archive. If it's a string, it is encoded as UTF-8 first.
- **compress\_type** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compression` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compress_type` field is also overridden.
- **compresslevel** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compresslevel` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compresslevel` field is also overridden.

**writestr\_distinfo**(*zinfo\_or\_arcname: Union[zipfile38.ZipInfo, str], data: Union[bytes, str], compress\_type: Optional[int] = None, compresslevel: Optional[int] = None*) → None

Write given data to the .dist-info directory.

This method is a handy shortcut for writing into `<dist>-<version>.dist-info/`, such that you don't have to generate the path yourself.

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

Does not permit writing into arcpaths of metadata files managed by this class.

#### Parameters

- **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 written into the archive. If it's a string, it is encoded as UTF-8 first.
- **compress\_type** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compression` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compress_type` field is also overridden.
- **compresslevel** – Same as in `zipfile.ZipFile.writestr`. Overrides the `compresslevel` parameter given to `__init__`. If the first parameter is a `ZipInfo` object, the value its `compresslevel` field is also overridden.

**Raises** `ProhibitedWriteError` – When attempting to write into `METADATA`, `WHEEL`, or `RECORD`.

## 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 algorithm 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**

- **arcpath** – Path in the archive of the file that the entry describes.
- **buf** – Buffer from which the data will be read in `HASH_BUF_SIZE` chunks. Must be fresh, i.e. `seek(0)`-ed.

**Raises** `RecordContainsDirectoryError` – If `arcpath` is a path to a directory.

**class** `wheelfile.WheelData`(\*, *generator: str = 'wheelfile 0.0.8', 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 0.0.8', 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_dist:
    Optional[List[str]] = None, requires_extensals: Optional[List[str]] = None,
    provides_extas: Optional[List[str]] = None, provides_dist: Optional[List[str]] =
    None, obsoletes_dist: 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 unchangeable. 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_dist`** – 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).





## 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.



## 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

### W

wheelfile, ??



## Symbols

[\\_\\_eq\\_\\_\(\) \(wheelfile.Metadata method\), 19](#)  
[\\_\\_eq\\_\\_\(\) \(wheelfile.WheelData method\), 16](#)  
[\\_\\_eq\\_\\_\(\) \(wheelfile.WheelRecord method\), 15](#)  
[\\_\\_init\\_\\_\(\) \(wheelfile.Metadata method\), 19](#)  
[\\_\\_init\\_\\_\(\) \(wheelfile.WheelData method\), 16](#)  
[\\_\\_init\\_\\_\(\) \(wheelfile.WheelFile method\), 6](#)  
[\\_\\_init\\_\\_\(\) \(wheelfile.WheelRecord method\), 15](#)  
[\\_\\_str\\_\\_\(\) \(wheelfile.Metadata method\), 19](#)  
[\\_\\_str\\_\\_\(\) \(wheelfile.WheelData method\), 16](#)  
[\\_\\_str\\_\\_\(\) \(wheelfile.WheelRecord method\), 15](#)  
[\\_\\_weakref\\_\\_ \(wheelfile.WheelFile attribute\), 9](#)  
[\\_\\_weakref\\_\\_ \(wheelfile.WheelRecord attribute\), 15](#)

## A

[abi\\_tag \(wheelfile.WheelFile attribute\), 6](#)

## B

[BadWheelFileError, 21](#)  
[build\\_tag \(wheelfile.WheelFile attribute\), 5](#)

## C

[close\(\) \(wheelfile.WheelFile method\), 9](#)  
[closed \(wheelfile.WheelFile attribute\), 6](#)

## D

[data\\_dirname \(wheelfile.WheelFile attribute\), 6](#)  
[distinfo\\_dirname \(wheelfile.WheelFile attribute\), 6](#)  
[distname \(wheelfile.WheelFile attribute\), 5](#)

## F

[filename \(wheelfile.WheelFile attribute\), 5](#)  
[from\\_wheelfile\(\) \(wheelfile.WheelFile class method\), 9](#)

## H

[hash\\_algo \(wheelfile.WheelRecord property\), 15](#)  
[hash\\_of\(\) \(wheelfile.WheelRecord method\), 15](#)

## I

[infolist\(\) \(wheelfile.WheelFile method\), 11](#)

## L

[language\\_tag \(wheelfile.WheelFile attribute\), 5](#)

## M

[Metadata \(class in wheelfile\), 16](#)  
[metadata \(wheelfile.WheelFile attribute\), 6](#)  
[module](#)  
     [wheelfile, 1](#)

## N

[namelist\(\) \(wheelfile.WheelFile method\), 11](#)

## P

[platform\\_tag \(wheelfile.WheelFile attribute\), 6](#)  
[ProhibitedWriteError, 21](#)

## R

[record \(wheelfile.WheelFile attribute\), 6](#)  
[resolved\(\) \(in module wheelfile\), 23](#)

## U

[UnnamedDistributionError, 21](#)  
[update\(\) \(wheelfile.WheelRecord method\), 15](#)

## V

[version \(wheelfile.WheelFile attribute\), 5](#)

## W

[WheelData \(class in wheelfile\), 15](#)  
[wheeldata \(wheelfile.WheelFile attribute\), 6](#)  
[wheelfile](#)  
     [module, 1](#)  
[WheelFile \(class in wheelfile\), 5](#)  
[WheelRecord \(class in wheelfile\), 15](#)  
[write\(\) \(wheelfile.WheelFile method\), 11](#)  
[write\\_data\(\) \(wheelfile.WheelFile method\), 12](#)  
[write\\_distinfo\(\) \(wheelfile.WheelFile method\), 12](#)  
[writestr\(\) \(wheelfile.WheelFile method\), 13](#)  
[writestr\\_data\(\) \(wheelfile.WheelFile method\), 13](#)  
[writestr\\_distinfo\(\) \(wheelfile.WheelFile method\), 14](#)