wheel Documentation

Release 0.29.0

Daniel Holth

Jul 20, 2018

Contents

User list | Dev list | Github | PyPI | User IRC: #pypa | Dev IRC: #pypa-dev

This library is the reference implementation of the Python wheel packaging standard, as defined in PEP 427.

It has two different roles:

1. A setuptools extension for building wheels that provides the `bdist_wheel` setuptools command
2. A command line tool for working with wheel files

CHAPTER 1

Quickstart

To build a wheel for your setuptools based project:

```
python setup.py bdist_wheel
```

The wheel will go to `dist/yourproject-<tags>.whl`.

If you want to make universal (Python 2/3 compatible, pure Python) wheels, add the following section to your `setup.cfg`:

```
[bdist_wheel]
universal = 1
```

To convert an `.egg` or file to a wheel:

```
wheel convert youreggfile.egg
```

Similarly, to convert a Windows installer (made using `python setup.py bdist_wininst`) to a wheel:

```
wheel convert yourinstaller.exe
```

CHAPTER 2

Installation

You can use pip to install wheel:

```
pip install wheel
```

If you do not have pip installed, see its documentation for installation instructions.

If you prefer using your system package manager to install Python packages, you can typically find the wheel package under one of the following package names:

- python-wheel
- python2-wheel
- python3-wheel

2.1 Python and OS Compatibility

wheel should work on any Python implementation and operating system and is compatible with Python version 2.7 and upwards of 3.4.

CHAPTER 3

User Guide

3.1 Building Wheels

Building wheels from a setuptools based project is simple:

```
python setup.py bdist_wheel
```

This will build any C extensions in the project and then package those and the pure Python code into a `.whl` file in the `dist` directory.

If your project contains no C extensions and is expected to work on both Python 2 and 3, you will want to tell wheel to produce universal wheels by adding this to your `setup.cfg` file:

```
[bdist_wheel]
universal = 1
```

3.2 Including license files in the generated wheel file

Several open source licenses require the license text to be included in every distributable artifact of the project. By default, `wheel` conveniently includes files matching the following glob patterns in the `.dist-info` directory:

- `LICEN[CS]E*`
- `COPYING`
- `NOTICE`

This can be overridden by setting the `license_files` option in the `[metadata]` section of the project's `setup.cfg`. For example:

```
[metadata]
license_files =
```

(continues on next page)

(continued from previous page)

```
    license.txt
    3rdparty/*.txt
```

No matter the path, all the matching license files are written in the wheel in the `.dist-info` directory based on their file name only.

By specifying an empty `license_files` option, you can disable this functionality entirely.

Note: There used to be an option called `license_file` (singular). As of wheel v1.0, this option has been deprecated in favor of the more versatile `license_files` option.

3.3 Converting Eggs to Wheels

The wheel tool is capable of converting eggs to the wheel format. It works on both `.egg` files and `.egg` directories, and you can convert multiple eggs with a single command:

```
wheel convert blah-1.2.3-py2.7.egg foo-2.0b1-py3.5.egg
```

The command supports wildcard expansion as well (via `iglob()`) to accommodate shells that do not do such expansion natively:

```
wheel convert *.egg
```

By default, the resulting wheels are written to the current working directory. This can be changed with the `--dest-dir` option:

```
wheel convert --dest-dir /tmp blah-1.2.3-py2.7.egg
```

3.4 Installing Wheels

Note: The `wheel install` command is merely a Proof-Of-Concept implementation and lacks many features provided by pip. It is meant only as an example for implementors of packaging tools. End users should use `pip install` instead.

To install a wheel file in `site-packages`:

```
$ wheel install someproject-1.5.0-py2-py3-none.whl
```

This will unpack the archive in your current site packages directory and install any console scripts contained in the wheel.

You can accomplish the same in two separate steps (with `<site-packages-dir>` being the path to your `site-packages` directory:

```
$ wheel unpack -d <site-packages-dir> someproject-X.Y.Z-py2-py3-none.whl
$ wheel install-scripts someproject
```

CHAPTER 4

Reference Guide

4.1 wheel convert

4.1.1 Usage

```
wheel convert [options] <egg_file_or_directory> [egg_file_or_directory...]
```

4.1.2 Description

Convert one or more eggs (`.egg`; made with `bdist_egg`) or Windows installers (`.exe`; made with `bdist_wininst`) into wheels.

Egg names must match the standard format:

- `<project>-<version>-pyX.Y` for pure Python wheels
- `<project>-<version>-pyX.Y-<arch>` for binary wheels

4.1.3 Options

`-d, --dest-dir` `<dir>`
Directory to store the generated wheels in (defaults to current directory).

4.1.4 Examples

- Convert a single egg file:

```
$ wheel convert foobar-1.2.3-py2.7.egg
$ ls *.whl
foobar-1.2.3-py27-none.whl
```

- If the egg file name is invalid:

```
$ wheel convert pycharm-debug.egg
"pycharm-debug.egg" is not a valid egg name (must match at least name-version-pyX.
↪Y.egg)
$ echo $?
1
```

4.2 wheel unpack

4.2.1 Usage

```
wheel unpack <wheel_file>
```

4.2.2 Description

Unpack the given wheel file.

This is the equivalent of `unzip <wheel_file>`, except that it also checks that the hashes and file sizes match with those in `RECORD` and exits with an error if it encounters a mismatch.

4.2.3 Options

-d, --dest-dir <dir>
Directory to unpack the wheel into.

4.2.4 Examples

- Unpack a wheel:

```
$ wheel unpack someproject-1.5.0-py2-py3-none.whl
Unpacking to: ./someproject-1.5.0
```

- If a file's hash does not match:

```
$ wheel unpack someproject-1.5.0-py2-py3-none.whl
Unpacking to: ./someproject-1.5.0
Traceback (most recent call last):
...
wheel.install.BadWheelFile: Bad hash for file 'mypackage/module.py'
$ echo $?
1
```

4.3 wheel repack

4.3.1 Usage

```
wheel repack <wheel_directory>
```

4.3.2 Description

Repack a previously unpacked wheel file.

This command can be used to repack a wheel file after its contents have been modified. This is the equivalent of `zip -r <wheel_file> <wheel_directory>` except that it regenerates the `RECORD` file which contains hashes of all included files.

4.3.3 Options

`-d, --dest-dir` `<dir>`
: Directory to put the new wheel file into.

4.3.4 Examples

- Unpack a wheel, add a dummy module and then repack it:

```
$ wheel unpack someproject-1.5.0-py2-py3-none.whl
Unpacking to: ./someproject-1.5.0
$ touch someproject-1.5.0/somepackage/module.py
$ wheel repack someproject-1.5.0
Repacking wheel as ./someproject-1.5.0-py2-py3-none.whl...OK
```

CHAPTER 5

Development

5.1 Pull Requests

- Submit Pull Requests against the *master* branch.
- Provide a good description of what you're doing and why.
- Provide tests that cover your changes and try to run the tests locally first.

Example. Assuming you set up GitHub account, forked wheel repository from https://github.com/pypa/wheel to your own page via web interface, and your fork is located at https://github.com/yourname/wheel

```
$ git clone git@github.com:pypa/wheel.git
$ cd wheel
# ...
$ git diff
$ git add <modified> ...
$ git status
$ git commit
```

You may reference relevant issues in commit messages (like #1259) to make GitHub link issues and commits together, and with phrase like "fixes #1259" you can even close relevant issues automatically. Now push the changes to your fork:

```
$ git push git@github.com:yourname/wheel.git
```

Open Pull Requests page at https://github.com/yourname/wheel/pulls and click "New pull request". That's it.

5.2 Automated Testing

All pull requests and merges to 'master' branch are tested in Travis based on our .travis.yml file.

Usually, a link to your specific travis build appears in pull requests, but if not, you can find it on our travis pull requests page.

The only way to trigger Travis to run again for a pull request is to submit another change to the pull branch.

5.3 Running Tests

Python requirements: tox or pytest

To run the tests locally:

```
$ tox                    # Runs the tests against all matching interpreters
$ tox -e py35            # Runs the tests against a specific environment
$ pip install -e .[test] # Installs the test dependencies locally
$ pytest                 # Runs the tests with the current interpreter
```

5.4 Getting Involved

The wheel project welcomes help in the following ways:

- Making Pull Requests for code, tests, or docs.
- Commenting on open issues and pull requests.
- Helping to answer questions on the mailing list.

5.5 Release Process

1. Make sure there is a version block for this release in `docs/news.rst` that mentions all the new user-facing changes
2. Add the version tag to the repository using `git tag X.Y.Z` (e.g. `git tag 1.0.1`)
3. Push the new tag to Github using `git push --tags`

When a new tag is pushed to Github, Travis will pick it up and automatically build the sdist and wheel and upload them to PyPI.

CHAPTER 6

Release Notes

UNRELEASED

- Removed wheel signing and verifying features
- Removed the "wheel install" and "wheel installscripts" commands
- Added the `wheel pack` command
- Allowed multiple license files to be specified using the `license_files` option
- Deprecated the `license_file` option
- Eliminated duplicate lines from generated requirements in `.dist-info/METADATA` (thanks to Wim Glenn for the contribution)

0.31.1

- Fixed arch as `None` when converting eggs to wheels

0.31.0

- Fixed displaying of errors on Python 3
- Fixed single digit versions in wheel files not being properly recognized
- Fixed wrong character encodings being used (instead of UTF-8) to read and write `RECORD` (this sometimes crashed bdist_wheel too)
- Enabled Zip64 support in wheels by default
- Metadata-Version is now 2.1
- Dropped DESCRIPTION.rst and metadata.json from the list of generated files
- Dropped support for the non-standard, undocumented `provides-extra` and `requires-dist` keywords in setup.cfg metadata
- Deprecated all wheel signing and signature verification commands
- Removed the (already defunct) `tool` extras from setup.py

0.30.0

- Added py-limited-api {cp32|cp33|cp34|...} flag to produce cpNN.abi3.{arch} tags on CPython 3.
- Documented the `license_file` metadata key
- Improved Python, abi tagging for `wheel convert`. Thanks Ales Erjavec.
- Fixed > being prepended to lines starting with "From" in the long description
- Added support for specifying a build number (as per PEP 427). Thanks Ian Cordasco.
- Made the order of files in generated ZIP files deterministic. Thanks Matthias Bach.
- Made the order of requirements in metadata deterministic. Thanks Chris Lamb.
- Fixed `wheel install` clobbering existing files
- Improved the error message when trying to verify an unsigned wheel file
- Removed support for Python 2.6, 3.2 and 3.3.

0.29.0

- Fix compression type of files in archive (Issue #155, Pull Request #62, thanks Xavier Fernandez)

0.28.0

- Fix file modes in archive (Issue #154)

0.27.0

- Support forcing a platform tag using `--plat-name` on pure-Python wheels, as well as nonstandard platform tags on non-pure wheels (Pull Request #60, Issue #144, thanks Andrés Díaz)
- Add SOABI tags to platform-specific wheels built for Python 2.X (Pull Request #55, Issue #63, Issue #101)
- Support reproducible wheel files, wheels that can be rebuilt and will hash to the same values as previous builds (Pull Request #52, Issue #143, thanks Barry Warsaw)
- Support for changes in keyring >= 8.0 (Pull Request #61, thanks Jason R. Coombs)
- Use the file context manager when checking if dependency_links.txt is empty, fixes problems building wheels under PyPy on Windows (Issue #150, thanks Cosimo Lupo)
- Don't attempt to (recursively) create a build directory ending with `..` (invalid on all platforms, but code was only executed on Windows) (Issue #91)
- Added the PyPA Code of Conduct (Pull Request #56)

0.26.0

- Fix multiple entrypoint comparison failure on Python 3 (Issue #148)

0.25.0

- Add Python 3.5 to tox configuration
- Deterministic (sorted) metadata
- Fix tagging for Python 3.5 compatibility
- Support py2-none-'arch' and py3-none-'arch' tags
- Treat data-only wheels as pure
- Write to temporary file and rename when using wheel install –force

0.24.0

- The python tag used for pure-python packages is now .pyN (major version only). This change actually occurred in 0.23.0 when the –python-tag option was added, but was not explicitly mentioned in the changelog then.

- wininst2wheel and egg2wheel removed. Use "wheel convert [archive]" instead.
- Wheel now supports setuptools style conditional requirements via the extras_require={} syntax. Separate 'extra' names from conditions using the : character. Wheel's own setup.py does this. (The empty-string extra is the same as install_requires.) These conditional requirements should work the same whether the package is installed by wheel or by setup.py.

0.23.0

- Compatibility tag flags added to the bdist_wheel command
- sdist should include files necessary for tests
- 'wheel convert' can now also convert unpacked eggs to wheel
- Rename pydist.json to metadata.json to avoid stepping on the PEP
- The –skip-scripts option has been removed, and not generating scripts is now the default. The option was a temporary approach until installers could generate scripts themselves. That is now the case with pip 1.5 and later. Note that using pip 1.4 to install a wheel without scripts will leave the installation without entry-point wrappers. The "wheel install-scripts" command can be used to generate the scripts in such cases.
- Thank you contributors

0.22.0

- Include entry_points.txt, scripts a.k.a. commands, in experimental pydist.json
- Improved test_requires parsing
- Python 2.6 fixes, "wheel version" command courtesy pombredanne

0.21.0

- Pregenerated scripts are the default again.
- "setup.py bdist_wheel –skip-scripts" turns them off.
- setuptools is no longer a listed requirement for the 'wheel' package. It is of course still required in order for bdist_wheel to work.
- "python -m wheel" avoids importing pkg_resources until it's necessary.

0.20.0

- No longer include console_scripts in wheels. Ordinary scripts (shell files, standalone Python files) are included as usual.
- Include new command "python -m wheel install-scripts [distribution [distribution ...]]" to install the console_scripts (setuptools-style scripts using pkg_resources) for a distribution.

0.19.0

- pymeta.json becomes pydist.json

0.18.0

- Python 3 Unicode improvements

0.17.0

- Support latest PEP-426 "pymeta.json" (json-format metadata)

0.16.0

- Python 2.6 compatibility bugfix (thanks John McFarlane)
- Bugfix for C-extension tags for CPython 3.3 (using SOABI)

- Bugfix for bdist_wininst converter "wheel convert"
- Bugfix for dists where "is pure" is None instead of True or False
- Python 3 fix for moving Unicode Description to metadata body
- Include rudimentary API documentation in Sphinx (thanks Kevin Horn)

0.15.0

- Various improvements

0.14.0

- Changed the signature format to better comply with the current JWS spec. Breaks all existing signatures.
- Include `wheel unsign` command to remove RECORD.jws from an archive.
- Put the description in the newly allowed payload section of PKG-INFO (METADATA) files.

0.13.0

- Use distutils instead of sysconfig to get installation paths; can install headers.
- Improve WheelFile() sort.
- Allow bootstrap installs without any pkg_resources.

0.12.0

- Unit test for wheel.tool.install

0.11.0

- API cleanup

0.10.3

- Scripts fixer fix

0.10.2

- Fix keygen

0.10.1

- Preserve attributes on install.

0.10.0

- Include a copy of pkg_resources. Wheel can now install into a virtualenv that does not have distribute (though most packages still require pkg_resources to actually work; wheel install distribute)
- Define a new setup.cfg section [wheel]. universal=1 will apply the py2.py3-none-any tag for pure python wheels.

0.9.7

- Only import dirspec when needed. dirspec is only needed to find the configuration for keygen/signing operations.

0.9.6

- requires-dist from setup.cfg overwrites any requirements from setup.py Care must be taken that the requirements are the same in both cases, or just always install from wheel.
- drop dirspec requirement on win32
- improved command line utility, adds 'wheel convert [egg or wininst]' to convert legacy binary formats to wheel

0.9.5

- Wheel's own wheel file can be executed by Python, and can install itself: `python wheel-0.9.5-py27-none-any/wheel install ...`
- Use argparse; basic `wheel install` command should run with only stdlib dependencies.
- Allow requires_dist in setup.cfg's [metadata] section. In addition to dependencies in setup.py, but will only be interpreted when installing from wheel, not from sdist. Can be qualified with environment markers.

0.9.4

- Fix wheel.signatures in sdist

0.9.3

- Integrated digital signatures support without C extensions.
- Integrated "wheel install" command (single package, no dependency resolution) including compatibility check.
- Support Python 3.3
- Use Metadata 1.3 (PEP 426)

0.9.2

- Automatic signing if WHEEL_TOOL points to the wheel binary
- Even more Python 3 fixes

0.9.1

- 'wheel sign' uses the keys generated by 'wheel keygen' (instead of generating a new key at random each time)
- Python 2/3 encoding/decoding fixes
- Run tests on Python 2.6 (without signature verification)

0.9

- Updated digital signatures scheme
- Python 3 support for digital signatures
- Always verify RECORD hashes on extract
- "wheel" command line tool to sign, verify, unpack wheel files

0.8

- none/any draft pep tags update
- improved wininst2wheel script
- doc changes and other improvements

0.7

- sort .dist-info at end of wheel archive
- Windows & Python 3 fixes from Paul Moore
- pep8
- scripts to convert wininst & egg to wheel

0.6

- require distribute >= 0.6.28
- stop using verlib

0.5

- working pretty well

0.4.2

- hyphenated name fix

0.4

- improve test coverage
- improve Windows compatibility
- include tox.ini courtesy of Marc Abramowitz
- draft hmac sha-256 signing function

0.3

- prototype egg2wheel conversion script

0.2

- Python 3 compatibility

0.1

- Initial version

Index

Symbols

C