PEP: Python Distribution Specification Documentation

Release 0.1

Donald Stufft

August 31, 2016

Contents

The goal of this PEP is to provide a standard layout and meta-data for Python distributions, so that all tools creating and installing distributions are inter-operable.

To achieve this goal this PEP proposes a new format for describing meta-data and layout of the distribution archive.

CHAPTER 1

Rationale

There are a number of problems currently in Python packaging.

- Lack of a standard cross tool layout for distributions.
- Multiple locations where the same meta-data is defined.
- Ability to build all types of projects.

1.1 Standard Layout

Right now there are a number of competing standards for what is contained inside of a distribution archive. distutils and setuptools share an idiom of using a `setup.py`, distutils2 uses a `setup.cfg`, and bento uses a `bento.info`.

This is further compounded by the fact that due to the executable nature of the distutils/setuptools standard `setup.py` distutils2 and bento can bootstrap themselves using code located inside of `setup.py`.

1.2 Meta-data

Currently meta-data can be located in one of a minimum of two locations. `PKG-INFO` and `setup.py`. It can also be located inside of `setup.cfg`, `bento.info`, and any other location that a packager might wish (again due to the executable nature of `setup.py`).

1.3 Custom Compilation

A number of projects have had to work around or monkeypatch distutils because of assumptions that distutils makes about how to compile a project were wrong. This includes projects that want to cross compile [1] and projects with complex compiler dependencies such as Numpy [2].

Further more there have been serious doubts raised by some that any generic compilation step would be able to cover all needs [3] [4].

[1] Cross-Compiling Python & C Extensions for Embedded Systems (http://pyvideo.org/video/682/cross-compiling-python-c-extensions-for-embedde)

[2] Packaging (numpy.distutils) (http://docs.scipy.org/doc/numpy/reference/distutils.html)

[3] Status of Packaging in 3.3 (http://mail.python.org/pipermail/python-dev/2012-June/120696.html)

[4] Status of Packaging in 3.3 (http://mail.python.org/pipermail/python-dev/2012-June/120591.html)

1.4 What this PEP proposes

- A new defined layout that any tool may create or consume
- A singular location to be used as the "one true source" for all meta-data
- New meta-data version to deal with new requirements.

CHAPTER 2

Standard Layout

All Python distributions are gzip archived containing a `dist.json` file as well as any source or binary files that should be included as part of the distribution.

2.1 Source Distribution

A source distribution is defined as a distribution that does not include any sort of precompiled files. A source distribution **MUST** contain a `dist.json` and all source files, Python or otherwise, that this distribution contains.

2.2 Binary Distribution

A binary distribution is defined as package that does not require any sort of compilation step to complete. A binary distribution **MUST** contain a `dist.json` as well as one or more directories containing a compiled distribution.

2.2.1 Build Directories

Build directories are specially named directories that signify which Platform and Python that particular build is for.

TODO: Specify the proper naming convention for build directories.

CHAPTER 3

dist.json

`dist.json` is a JSON file containing all the meta-data for this distribution. It must be a valid JSON file and cannot be a JavaScript object literal.

3.1 Fields

3.1.1 name

The most important things in your dist.json are the name and version fields. The name and version together form an identifier that is assumed to be completely unique. Changes to the distribution should come along with changes to the version.

The name is what your thing is called.

3.1.2 version

The most important things in your dist.json are the name and version fields. The name and version together form an identifier that is assumed to be completely unique. Changes to the distribution should come along with changes to the version.

The version must be in the format specified in PEP 386 [7]

3.1.3 summary

A one-line summary of what the distribution does.

3.1.4 description

A longer description of the distribution that can run to several paragraphs. Software that deals with metadata should not assume any maximum size for this field, though people shouldn't include their instruction manual as the description.

The contents of this field can be written using reStructuredText markup [#rest]. For programs that work with the metadata, supporting markup is optional; programs can also display the contents of the field as-is. This means that authors should be conservative in the markup they use.

[7] PEP 386 - Changing the version comparison module in Distutils (http://www.python.org/dev/peps/pep-0386)

3.1.5 keywords

A list of additional keywords to be used to assist searching for the distribution in a larger catalog. It should be a list of strings.

Example:

```
{
    "keywords": ["dog", "puppy", "voting election"],
}
```

3.1.6 author

A string, dictionary representing the author of the distribution, see *People Fields* for more information.

3.1.7 maintainer

A string or dictionary representing the current maintainer of the distribution, see *People Fields* for more information. This field **SHOULD** be omitted if it is the same as the author.

3.1.8 contributors

A list of additional contributors for the distribution. Each item in the list must either be a string or a dictionary, see *People Fields* for more information.

3.1.9 uris

A dictionary of Label: URI for this project. Each label is limited to 32 characters in length.

Example:

```
{
    "uris": {
        "Home Page": "http://python.org/",
        "Bug Tracker": "http://bugs.python.org/"
    }
}
```

3.1.10 license

Text indicating the license covering the distribution where the license is not a selection from the "License" Trove classifiers. See *classifiers* below. This field may also be used to specify a particular version of a license which is named via the Classifier field, or to indicate a variation or exception to such a license.

3.1.11 classifiers

A List of strings where each item represents a distinct classifier for this distribution. Classifiers are described in PEP 301 [6].

Example:

[6] PEP 301 - Package Index and Metadata for Distutils (http://www.python.org/dev/peps/pep-0301/)

```
{
    "classifiers": [
        "Development Status :: 4 - Beta",
        "Environment :: Console (Text Based)"
    ]
}
```

3.1.12 platform

A Platform specification describing an operating system supported by the distribution which is not listed in the "Operating System" Trove *classifiers*.

3.1.13 requires_python

This field specifies the Python version(s) that the distribution is guaranteed to be compatible with. Version numbers must be in the format specified in *Version Specifiers*.

3.2 People Fields

The `author`, and `maintainer` fields, and the `contributors` field list items each accept either a string or a dictionary. The dictionary is a mapping of `name`, `email`, and `url`, like this:

```
{
    "name": "Monty Python",
    "email": "monty@python.org",
    "url": "http://python.org/"
}
```

Any of the fields may be omitted where they are unknown. Additionally they may be specified using a string in the format of `Name <email> (url)`. An example:

```
Monty Python <monty@python.org> (http://python.org/)
```

3.3 Version Specifiers

Version specifiers are a series of conditional operators and version numbers, separated by commas. Conditional operators must be one of "<", ">", "<=", ">=", "==", "!=" and "~>".

The "~>" is a special case which can be pronounced as "approximately greater than". When this is used it signifies that the the version should be greater than or equal to the specified version within the same release series. For example, if "~>2.5.2" is the specifier, then any version matching 2.5.x will be accepted where x is >= 2.

Any number of conditional operators can be specified, e.g. the string ">1.0, !=1.3.4, <2.0" is a legal version declaration. The comma (",") is equivalent to the **and** operator.

Each version number must be in the format specified in PEP 386 [7].

Notice that some projects might omit the ".0" prefix for the first release of the "2.5.x" series:

- 2.5
- 2.5.1

- 2.5.2
- etc.

In that case, “2.5.0” will have to be explicitly used to avoid any confusion between the “2.5” notation that represents the full range. It is a recommended practice to use schemes of the same length for a series to completely avoid this problem.

CHAPTER 4

References

CHAPTER 5

Copyright