Packaging for Forest

Josh Barback

August 27, 2020

Overview

1. Advantages of packaging
2. The “regular” package framework
3. Namespace packages
4. Native/implicit namespace package template
5. `setup.py` for a namespace package
6. Suggested style for imports in a namespace package module
7. Resources

1. Advantages of packaging

- In the context of Python code, packaging is an easy way to:
  - Create an organized framework for development & testing,
  - Attach metadata such as licensing & documentation,
  - Assert dependencies,
  - Implement software versioning,
  - Distribute code, e.g. via PyPI.

- Benefits for end-users:
  - Packaged code is easy to import,
  - Packaging enables version-to-version consistency.

2. The “regular” package framework

```
package
    LICENSE.md
    README.md
    setup.py ← Instructions for installing this package
    package
        __init__.py ← This module runs when the package is imported
        module1.py
        module2.py
        module3.py
        noncode1.csv
        noncode2.json
```

3. Namespace packages

- A namespace package bundles multiple sub-packages. Each sub-package can be imported separately, e.g. `import package.subpackage`.
- From PyPA: *Namespace packages can be useful for a large collection of loosely-related packages (such as a large corpus of client libraries for multiple products from a single company).*
- Many packages use some type of namespace framework, e.g. `scipy`, `pandas`.
- Python 3.3+ supports a “native” namespace packaging framework that is easy to implement.

4. Native/implicit namespace package template

```
package
    LICENSE.md
    README.md
    setup.py ← How to install this package
    package
        subpackageA
            __init__.py ← Runs when subpackageA is imported
            moduleA1.py
            moduleA2.py
            noncodeA1.csv
            noncodeA2.json
        subpackageB
            __init__.py ← Runs when subpackageB is imported
            moduleB.py
        subpackageC
            __init__.py ← Runs when subpackageC is imported
            moduleC.py
```

5. setup.py for a namespace package

```
from setuptools import setup, find_namespace_packages ← Import installation functions

requires = ['numpy', 'pandas'] ← Dependencies to install

with open('README.md') as f: ← Load the README
    readme = f.read()

with open('LICENSE.md') as f: ← Load the LICENSE
    license = f.read()

setup(
    name='package_name',
    version='0.0.1',
    description='Description of the package',
    long_description=readme,
    author='author name',
    author_email='address@domain',
    license=license,
    packages=find_namespace_packages(include=['package.*']), ← Which directories are subpackages
    package_data={'': ['*.json']}, ← Specify non-code files to install
    install_requires=requires
)
```

6. Suggested style for imports in a namespace package module

`import logging` ← Start with imports from the Standard Library
`import numpy` ← Import common packages
`import timezonefinder` ← Import specialized packages
`import package.subpackage` ← Use absolute imports for other subpackages in the same package

`import .module` ← Finally, use relative imports for modules from the same subpackage

7. Resources

- **Important links:**

  The Python Package Index

  PyPA's Python Packaging User Guide

- **Some relevant PEPs:**

  PEP 420 – Implicit Namespace Packages

  PEP 423 – Naming conventions and recipes related to packaging

- **Useful articles:**

  The Joy of Packaging

  "Structuring Your Project" from *The Hitchhiker's Guide to Python*