PYTHON PACKAGING

Contact Information

Nick Wilson nick@njwilson.net
Ryan Baker bakerry@engr.oregonstate.edu
Taylor Hand handt@engr.oregonstate.edu
Taylor Michael michaeta@engr.oregonstate.edu

http://osupython.wordpress.com/

INFORMATION

There are currently multiple methods for packaging, distributing, discovering, and installing Python software.

Addressed in the poster are the limitations of the current methods, the efforts to standardize Python packaging, and advice for developers to prepare for to add support for the new packaging library, distutils2 ("packaging" in Python 3.3+), to their software.

Packaging code into modules allows one's code to be searchable, installable, uninstallable, and provides management for dependencies.

Packages allow code to be easily distributed and updated by end-users.

PAST / PRESENT

Packaging in Python began in the 90's with the need of some method to bundle and distribute packages in a way similar to Perl's CPAN. This lead to a special intrest group (SIG) started in 1998 which began work on Distutils.

Distutils was introduced into the Python standard library in 2000. It addressed basic packaging issues such as tarballing and installing packages. While it provided much needed basic packaging features, it lacks advanced features we have come to expect with modern package managers. This lead to the development of setuptools by Phillip Eby.

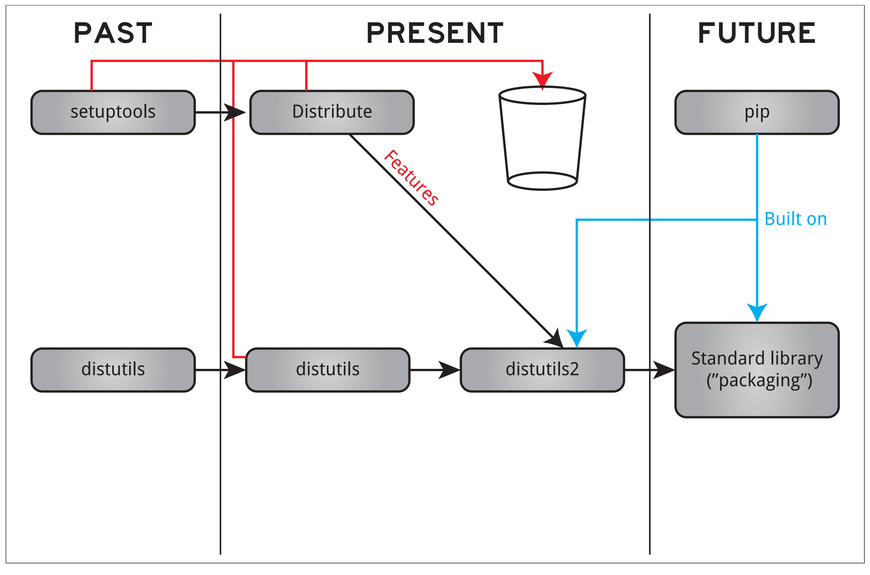
Setuptools is a set of extentions on top of distutils that incorporates a lot of needed features such as binary installation and automatic installation of dependencies. Phillip began work on setuptools in 2004 and had a full featured version of it ready in 2005.

In 2008, the Distribute library (a fork of setuptools) began with the goal of improving packaging further by adding things such as Python 3 support, uninstallation of packages, and complying with various PEP's that had come about since the introduction of setuptools. Development has slowed since the introduction of distutils2.

Also in 2008, Ian Bicking introduced a tool named pip which also layered on top of setuptools with the goal of improving some of the faults with setuptools' easy_install command. Features such as uninstallation of packages and searching of PyPI were included in pip.

Currently, there are a number of options for handling packaging in python. The pip tool has become popular and provides support for using either setuptools or distribute underneath. Due to setuptools and distribute both being 3rd party packages, they need to be installed manually before pip can be used. Distutils is still included in Python standard library as a more basic package manager if distribute or setuptools are unavailable.

While they all attempt to maintain basic compatibility, the existance of multiple different packaging libraries somewhat fragments and slows development of packaging standards and best-practices among Python developers.



FUTURE

The newest advancements in Python packaging are taking place in Distutils2.

Distutils2 is a new distutils library set for inclusion in Python 3.3+, started as a fork of the distutils codebase, with good ideas taken from setuptools and thoroughly discussed in PEPs for some of them, and a basic installer inspired by pip.

The actual name that can be imported is "packaging" in the Python 3.3+ standard library or "distutils2" in 2.4+ and 3.1-3.2.

Distutils2 will provide its own installer, pysetup, but it will not obsolete pip. pysetup is intended to just provide a minimum set of features to get people started.

Developers can prepare for the new standard library by following a few guidelines when creating a new package:

- Use PEP 386 compatible versioning
- Make setup.py as simple as possible
- Have setup.py be functionally independent of the current installer (setuptools, ez_setup, etc); don't assume which installer will be used
- Submit only stable releases to pypi
- Handle data files carefully

- **Standard database of installed distributions (PEP 376)**
  - Facilitates interoperability between package managers
  - Supports uninstallation
- **Improved metadata (PEP 345)**
  - Dependencies on Python versions (e.g., >=2.5, <2.7)
  - Dependencies on distributions (i.e., name on PyPI) rather than modules
  - Static declaration of dependencies based on operating system, platform, and Python version
- **Standard distribution versioning scheme (PEP 386)**
  - Versions can be easily compared
  - Clear distinction between development, pre- and post-release, and final versions
- **New packaging library: packaging/distutils2**
  - Obsoletes distutils, setuptools, and distribute
  - Static configuration file (setup.cfg) replaces setup.py
  - Can be used as a library

NEW PACKAGING STANDARDS

PEP 345

Metadata for Python Software Packages 1.2

Goal

Define new metadata for distributions to better specify dependencies and relationships with other distributions.

Motivation

Previous standards specified dependencies on Python modules, but people don't distribute modules; they distribute distributions.

setuptools and distribute support dependencies on distributions (via install_requires), but it's not standardized.

Current methods require running code in setup.py to calculate dependencies based on operating system, platform, and Python version/implementation.

PEP 376

Database of Installed Python Distributions

Goal

Define a standard database of installed Python distributions so package managers are interoperable.

Motivation

Existing tools (distutils, setuptools/distribute/easy_install, and pip) each have their own method of tracking installed distributions.

Most of the existing tools don't support uninstallation.

No common API exists for querying installed distributions.

PEP 386

Changing the version comparison module in Distutils

Goal

Specify a common standard for version numbers.

Motivation

Without a common standard, installers (e.g., pip) have a hard time knowing which version of a package to install.

Pseudo-format

```
N.N[.N]+[{a|b|c|rc}N[.N]+][.postN][.devN]
```

NEW PACKAGING LIBRARY

packaging/distutils2

Goal

Provide a new standard packaging library that implements the new standards and provides additional improvements over the current libraries.

Motivation

setuptools made some valuable improvements, but it is still built on top of the flawed design of distutils.

Redesigning distutils would break all the current tools; a new library was needed to implement the new standards.

TOOL SUPPORT

pysetup	Barebones command-line tool shipped with packaging/distutils2
pip	Will be updated to use packaging/distutils2 instead of setuptools Continue to provide additional functionality (SCM support, requirements files, etc.)