Distribution using `distutils`

Anna Ravenscroft

PyCon 2004

Who are you

- Novice Pythonistas
  - Can program in Python
  - Can import modules
- New to Packaging
  - First time
  - Review fundamentals

Who am I

- Novice Pythonista
  - Background in Training and Office apps
  - Not a developer
  - Program in Python only
  - Technical editing
  - Tutor list and c.l.py
- Packaging
  - Questions on tutor and c.l.py
  - My project
  - Current documentation

Focus

- Simple stuff: just enough to get started
- Cross-platform, pure distributions
- Package `distutils` – standard library

What We'll Cover

- General considerations
- Directory Issues
- User-friendly `setup.py`
- Running `sdist`
- `MANIFEST.in`
- When to use `bdist`
- Alternatives to `distutils`

General Considerations

- What is packaging
- Why a distribution
- What goes in a distribution
- Kinds of distributions

What is Packaging

- A [Python] **Package** *(noun)*:
  - *A module that contains other modules*
- To **package** *(verb)*:

"A distribution is the set of files to package into a single file for distribution purposes."

--Python in a Nutshell

  - Self-installing executable (.exe, .rpm)
  - Archive – unpack and install (.zip, .tgz)

Why a Distribution

- Easy installation for user
- Module and version dependencies
- Installation Options
  - Default configuration
  - User's configuration
- C-extensions

What is IN a Distribution

- Packages (0, 1, more)
- Modules
- Python Scripts
- Other files:
  - data files
  - metadata about the distribution itself
  - C-coded extensions
  - ...

Pure and Non-pure Distributions

- *Pure:* Python only
- *Non-pure:* contains non-Python code
  - C-coded extensions
  - Pyrex
  - C++ or Fortran may be supported

Preparing your distribution

- Distribution root directory
- `setup.py`
- `README` or `README.txt`
- `MANIFEST.in`
- `setup.cfg`

Directory Issues

- All files in *Distribution root directory*
  - `setup.py`, `README` or `README.txt`
  - `MANIFEST.in`, `setup.cfg`
  - Scripts and top-level modules
- Subdirectories of the distribution root
  - `test/test*.py`
- Any packages:
  - Own package-specific subdirectory
  - Sub-directory name == package name
  - Include `__init__.py` in package subdir

User-friendly setup.py

```
from distutils.core import setup [, Extension]

setup(keyword arguments, )
```

- Import statements
- Call `setup` with keyword arguments
  - Metadata arguments
  - Content arguments

`setup.py` *import statements*

```
from distutils.core import setup [, Extension]
```

- Always import `setup`
- Only import `Extension` if needed for non-pure distribution

Metadata arguments

- Arguments must be strings
  - Literals, variables or read from a file
  - Exception: `keywords` list
- Required arguments
  - `name` of distribution, abbreviated if nec.
  - `url` for information about distribution

Optional Metadata arguments

- Optional but recommended
  - `author` not required, but don't you want the credit for this killer app?
  - `author_email` (**reqd** if author provided)
  - `description` <=80 chars
  - `fullname` if name abbreviated
  - `license` concise terms, refer to file/url
  - `version` (*major.minor*)

More metadata arguments

- Optional but useful
  - contact if different from author
  - contact_email (**reqd** if contact provided)
  - maintainer if different from author
  - maintainer_email (**reqd** if maintainer)
  - keywords ***list***: indexing in search engines
  - platforms if nec., concise, refer to file/url

Content arguments

- File paths must be relative to distribution root directory
  - Use / as path separator
- No paths for `packages`, `py_modules`
  - `setup` looks in distribution root directory
  - sub-packages must be explicitly specified
  - sub-packages use dot-name syntax

Content: packages, modules, scripts

```
packages=['pk1','pk1.sub1','pk2']

py_modules=['mod1', 'mod2',]

scripts=[“script1.py”,]
```

- Run as main programs, command line
- #! line, adjusted to point to interpreter

Content: data_files

```
data_files=[('imgs',['i1.jpg','/gfs/i2.gif']),]
```

- Target directory: where `distutils` places data files when installing
  - Subdir of Python's `sys.prefix` (pure)
  - Subdir of `sys.exec_prefix` (non-pure)
- Files put into flat target directory
  - **not** into subdirectories of the target
- Multiple (*directory*,[*files*]) pairs

Content: C-coded extensions

```
ext_modules=[ Extension('x ',sources=['x.c']) ]
```

- list of instances of class `Extension`
  - each with a name and a list of C source files from which the extension is built
- `distutils` compiles and links everything properly to build `x.so` (Linux), `x.pyd` (Windows), whatever
- **oodles** of options (for C hackers...:-)
- Reminder: must import `Extension`

setup.py example

```
from distutils.core import setup

myname=”Anna Ravenscroft”
myeddress=”anna@aleax.it”
distname=”anna_ex”
myfullname=”Anna's Packaging Example”
distvers=”0.1”
dist_url=”http://www.pycon.org/index.html”

setup(author=myname, author_email=myeddress,
   name=distname, fullname=myfullname,
   version=distvers, url=dist_url,
    scripts=[“hello_pkg.py”])
```

Running **sdist**

```
$ python setup.py sdist
running sdist
warning: sdist: manifest template 'MANIFEST.in' does
    not exist       (using default file list)
writing MANIFEST file 'MANIFEST'
creating Hello_World-0.1
making hard links in Hello_World-0.1...
hard linking setup.py -> Hello_World-0.1
tar –cf dist/Hello_World-0.1.tar Hello_World-0.1
gzip –f9 dist/Hello_World-0.1.tar
removing 'Hello_World-0.1' (and everything under it)
```

`sdist` *Default File List*

- Source files in `setup.py` arguments:
  - package, py_modules, extensions
- Files: `README.txt, setup.py, setup.cfg`
- Test files in `test/test*.py` in dist. root
- **not** scripts, **not** data_files
- Creates `MANIFEST` log
  - What got put where

Ins and Outs of `MANIFEST.in`

- Manifest template
- Text file
- Each line is a rule
  - Include files matching pattern: `include`
  - Include dir with `graft`
  - Exclude files matching pattern: `exclude`
  - Exclude dir with `prune`
- Normally starts with standard list
- Order matters

Example of `MANIFEST.in`

```
include *.py
```
include scripts in distribution root dir

```
recursive-include miscdata *.txt
```
include all text files anywhere under miscdata dir

```
graft images
```
include all files in subdir images/

```
exclude *.bmp
```
but not bitmaps

When to use `bdist`

- Minor convenience for pure distros
  - Use to create RPMs
  - Just need commands (`zip`, `tar`...) to produce compressed archive-file
- Non-pure distributions - advisable
  - appropriate C compiler
  - specific Python version
  - easiest to do **on** target platform

Alternatives to `distutils`

- `py2exe` (Windows)
  - Extension to `distutils`
  - Import in `setup.py`
  - Use like `sdist` or `bdist`

```
c:\foo>python setup.py py2exe
```

- `BuildApplet/BundleBuilder` (Mac)
- McMillan's `Installer` (Windows, Linux, standalone executables)

Building GUI Installers

- Inno Setup (free, Windows)

http://www.jrsoftware.org/isinfo.php

- Wise (commercial, Windows)

http://www.wise.com/

- InstallShield (commercial, multi-platform)

http://www.installshield.com/products/author.asp

Resources

- *Python in a Nutshell*
  - Alex Martelli (O'Reilly)
  - Ch. 26: Distributing Extensions and Programs
- *Distributing Python Modules*
  - by Greg Ward
  - For packagers
  - http://www.python.org/doc/current/dist/dist.html
- *Installing Python Modules*
  - by Greg Ward
  - For end-users/sys-admins
  - http://www.python.org/doc/current/inst/inst.html