Skip to content

Skip to search - Accesskey = s

James Gardner

The Pages

- About

The Search

search site archives

Submit

The Associates

- Andrew Chapman
- Ben Bangert
- Ian Bicking
- Jason Kottke
- Signal vs Noise
- Simon Willison
- Vecosys

The Storage

- December 2007
- November 2007
- October 2007
- September 2007
- August 2007
- July 2007
- June 2007
- May 2007
- April 2007
- March 2007
- July 2006
- May 2005

The Categories

- 3aims
- AuthKit
- Business
- Debian
- Desktop Software
- EC2
- Flash
- Hardware
- Hosting
- JavaScript
- Mako
- OpenID
- Physics
- PostgreSQL
- Pylons
- Python
- Software Releases
- sql
- SQLAlchemy
- Talks and Conferences
- ToscaWidgets
- Uncategorized
- Virtulization
- Web

The Meta

- Login
- WP
- XFN
- RSS
- Comments RSS
- Back to top

Creating a Python Package Using Eggs and Subversion

Posted in Pylons, Python, SQLAlchemy by thejimmyg on the November 8th, 2007

This blog post just documents one way of creating a package. This is for one named `SQLAlchemyManager` which is some experimental SQLAlchemy middleware I'm developing. Package names are normally lowercase but the project itself can use CamelCase so in this example the actual package is called `sqlalchemymanager`.

First create the directory structure:

```
mkdir SQLAlchemyManager
cd SQLAlchemyManager
mkdir docs
mkdir sqlalchemymanager
mkdir examples
mkdir tests
```

This tutorial assumes you are creating the new structure in an existing SVN tree. If you aren't you should now import the directories into a new Subversion repository.

Then create the `setup.cfg` file:

```
[egg_info]
tag_build = dev
tag_svn_revision = true
```

These options tell setuptools that this is a development release, and that the SVN revision number should be included in the package version. This will create an egg with `dev-r139.egg` or similar at the end of the filename. When you want to make a production release you should comment out these lines.

Next you need to set up `ez_setup.py` so that if your project's users don't have `setuptools` it will be automatically installed when they try to use it. You can do this svn:externals definition so that subversion will automatically include the latest files you need:

```
ez_setup svn://svn.eby-sarna.com/svnroot/ez_setup
```

You can set this by executing this command in your project directory:

```
svn propedit svn:externals .
```

And then adding the line shown above to the file that comes up for editing. Then, whenever you update your project, `ez_setup` will be updated as well.

Next you need a `setup.py` file which contains most of the metadata about the project. This one is setup to make use of the `ez_setup.py` you just set up if the user doesn't have setup tools. Importing `ez_setup` in this way doesn't cause problems with Buildout either:

```
try:
    from setuptools import setup, find_packages
except ImportError:
    from ez_setup import use_setuptools
    use_setuptools()
    from setuptools import setup, find_packages

import sys, os

version = '0.0'

setup(
    name='SQLAlchemyManager',
    version=version,
    description="",
    long_description="""\
    """,
    # Get strings from http://pypi.python.org/pypi?%3Aaction=list_classifiers
    classifiers=[],
    keywords='',
    author='',
    author_email='',
    url='',
    license='',
    packages=find_packages(exclude=['ez_setup', 'examples', 'tests']),
    include_package_data=True,
    zip_safe=False,
    install_requires=[
    ],
    entry_points="""
    """,
)
```

You can enter information for all the fields listed above but `version`, `install_requires` and `long_description` are particularly interesting.

`version`
: This shoud be a string containing the version number of the package in three parts. The first part is the maor version, the second part the minor version and the third part the revision. In this case I'm developing towards a 1.0 release so the major relase will be 0 until the software is stable enough for 1.0. This is the first version so the minor version is 1 and I haven't released any revisions so I'll call that 0. The version string is therefore `"0.1.0"`.

`install_requires`
: This should be a comma separated list of packages and their version numbers for all the packages the package you require depend on. SQLAlchemyManager depends on SQLAlchemy 0.4 so I'd add `"SQLAlchemy>=0.4,<=0.4.99"` to the install list. Notice that I specify `<=0.4.99`. The idea behind choosing this is that any incompatible changes to SQLAlchemy should result in a new minor version of SQLAlchemy being released so if my package works with 0.4 it should also work with 0.4.5, 0.4.9 and even 0.4.99 if there are that many revisions. It might not work with 0.5 though. You might think I could specify `<0.5` here but `0.5a` `0.5dev`" and `0.5rc1` are all treated as less that `0.5` by easy install so the recommendation is to use the `.99` format instead. You could choose `<=0.4.99999` if you wanted but pacjages rarely have more than about 30 revisions so 99 is virtually always fine.

`long_description`
: This is handy because it allow you to write detailed information in reStructuredText format. This information then forms the main text for the page which will appear on the Cheeseshop which effectively means you don't need to create a website for your project because you can keep all information in the long description. We'll see an example of a long description at the end.

Now you need to write some documentation. I create all the docs with a `.txt` extension and using Windows line endings so that the information can be easily read by both Windows and Linux users.

First create the `README.txt` file:

```
See the ``docs/index.txt`` for information.
```

Then the `CHANGELOG.txt` file:

```
Changes
=======

0.1.0 (**svn**)

* First version
```

Every time you make a change you should add it to the change log, I put the most recent change at the top. Every time you create a new version you should add a new section. The `(**svn**)` label marks which is the version in subversion. You should remember to move this if you create a new version or remove it if you create a production release.

Now the documentation in `docs/index.txt`:

```
SQLAlchemyManager
+++++++++++++++++++++++

.. contents ::

Summary
=======

* Provides a sensible way of using SQLAlchemy in WSGI applications

Get Started
===========

* Download from the `SQLAlchemyManager Cheeseshop page <http://python.org/pypi/SQLAlchemyManager>`_
* Install with ``easy_install SQLAlchemyManager``.

Author
======

`James Gardner <http://jimmyg.org/>`_ james at pythonweb dot org
```

You can now create the long description. Here's one way to do it. At the top of the `setup.py` file you can do this:

```
def read(*rnames):
    return open(os.path.join(os.path.dirname(__file__), *rnames)).read()

long_description = (
    "\n"+read('docs/index.txt')
    + '\n'
    + read('CHANGELOG.txt')
    + '\n'
    'License\n'
    '=======\n'
    + read('LICENSE.txt')
    + '\n'
    'Download\n'
    '========\n'
)
```

Then update ensure `long_description=long_description` in the `setup()` function. This will then automatically create the long description from the files in your project, helping to make life easier.

You can now create the files your project will use in the `sqlalchemy` directory and check them all in.

Finally you can add the `LICENSE.txt` file, preferably using one of the Open Source Licenses, the MIT license being the one I generally use. You should update the license line in `setup.py` too.

You'll also want to specify a summary which will appear in the index page on the cheeseshop. Choose a few words to describe the project and put them in the `description` argument string.

Making a Release

To make a release first export the code. You don't want to use the checkout in case the `.svn` files get included. You should then add the exported files as a tag in your subversion tree. Using this process rather than a branch will also mean that your distibution packages will contain the `ez_setup.py` at the time the export was done, rather than always using the most recent version.

Here's how to create a 0.1.0 release:

```
svn co http://somedomain/svn/SQLAlchemyManager/tags tags
svn export http://somedomain/svn/SQLAlchemyManager/trunk 0.1.0
mv 0.1.0 tags/
svn add tags/0.1.0
svn ci tags/0.1.0 -m "Creating the 0.1.0 tag"
```

Now edit the `CHANGELOG.txt` to remove the (**svn**) text. Update the version number in `setup.py` and comment out the `tag_build = dev` and `tag_svn_revision = true` lines in `setup.cfg`. Check in the changes and you are nearly there:

```
svn ci tags/0.1.0 -m "Final changes for the 0.1.0 release"
```

Now you can create the release:

```
svn export http://somedomain/svn/SQLAlchemyManager/tags/0.1.0 0.1.0
cd 0.1.0
python setup.py sdist bdist_egg register upload
```

This will create binary and source distributions for your package and upload them to the Cheeseshop.

In a future article I'll try to talk about documentation generation, development releases and unit testing.

8 Responses to 'Creating a Python Package Using Eggs and Subversion'

Subscribe to comments with RSS or TrackBack to 'Creating a Python Package Using Eggs and Subversion'.

1. Linux Code and More » Blog Archive » James Gardner: Creating a Python Package Using Eggs and Subversion said,

   on November 8th, 2007 at 1:20 pm

   [...] Linux :: Open Sorce with Linux wrote an interesting post today onHere's a quick excerpt...txt ... extension and using Windows line endings so that the information can be easily read by both Windows and Linux users. First create the [...]

2. Linux Code and More » Blog Archive » Creating a Python Package Using Eggs and Subversion said,

   on November 8th, 2007 at 1:49 pm

   [...] LXer: Linux News wrote an interesting post today onHere's a quick excerpt...txt ... extension and using Windows line endings so that the information can be easily read by both Windows and Linux users. First create the [...]

3. Chris said,

   on November 8th, 2007 at 2:51 pm

   great article, really helpful!

4. see whatever... » Eggs and SVN said,

   on November 8th, 2007 at 2:53 pm

   [...] Creating a Python Package Using Eggs and Subversion [...]

5. Chris said,

   on November 8th, 2007 at 4:02 pm

   Isn't using 'svn cp' easier to use, and more efficient?

   svn cp http://somedomain/svn/SQLAlchemyManager/trunk http://somedomain/svn/SQLAlchemyManager/tags/0.1.0

   Any changes to trunk's files won't affect the files in this branch.

6. James Gardner » Creating a Python Package Using Eggs and Subversion said,

   on November 8th, 2007 at 4:58 pm

   [...] iwallace wrote an interesting post today onHere's a quick excerpt [...]

7. joe said,

   on November 9th, 2007 at 12:53 pm

   does cheeseshop host the files ?
   do you have to include sources or can you just deliver pyc ?

8. Patrick said,

   on November 9th, 2007 at 1:11 pm

   Nice one dude... i may have a crack at that... also call me sometime ur phone is not working.. or u have a new number

Leave a Reply

Name (required)

Mail (will not be published) (required)

Website

Submit Comment
