Generating data files in setup.py

By Petri Lehtinen on 2011-01-28

In a work project, I have a few JavaScript files that are generated from a bunch of other files. The project is a Django website, so I just have views that generate the files on-the-fly when running in debug mode, and everything works nice and smooth.

For production, though, I needed the flat files that would be served from disk. I figured out that the best approach would be to generate the files in `setup.py` upon installation, but I could only find very superficial documentation on how to do that.

A brief intro to `setup.py`: In every project's `setup.py` file, the `setup` function, from Python standard library's `distutils.core` module, is used to define the project's files and metadata. (`setup` can also be imported from from setuptools or distribute, but they're compatible with distutils.) With the standard commands that `setup.py` provides, the files and metadata can be compiled to an egg, distributed as a source tarball, uploaded to PyPI, and so on.

The entry point to altering `setup.py`'s behaviour is the optional `cmdclass` argument to the `setup` function. It's value is a `dict` from command names to `distutils.command.Command` subclasses that implement the commands. The `build_py` command is where the package data files are installed, so to override `build_py`, I created the class `my_build_py` and registered it, like this:

```
from distutils.core import setup
from distutils.command.build_py import build_py

class my_build_py(build_py):
    # ...

setup(
    # Define metadata, files, etc.
    # ...
    cmdclass={'build_py': my_build_py}
```

```
)
```

The `run` method of `build_py`, along with copying and compiling the Python source files, is responsible for copying the packages data files to the build directory `build/lib.<platform>`. (The actual directory name is stored in the `build_py` instance's `self.build_lib` variable.)

To install your own files, just override the `run` method. Remember to call the superclass after you're done with your own files.

```
def generate_content():
    # generate the file content...
    return content

class my_build_py(build_py):
    def run(self):
        # honor the --dry-run flag
        if not self.dry_run:
            target_dir = os.path.join(self.build_lib, 'mypkg/media')

            # mkpath is a distutils helper to create directories
            self.mkpath(target_dir)

            with open(os.path.join(target_dir, 'myfile.js'), 'w'):
                fobj.write(generate_content())

        # distutils uses old-style classes, so no super()
        build_py.run(self)
```

And that's it! A later phase of the installation copies everything from `build/lib.<platform>` to the correct place, so your generated file gets in, too.

Tags: python distutils