Freezing Your Code

“Freezing” your code is creating a single-file executable file to distribute to end-users, that contains all of your application code as well as the Python interpreter.

Applications such as ‘Dropbox’, ‘Eve Online’, ‘Civilization IV’, and BitTorrent clients do this.

The advantage of distributing this way is that your application will “just work”, even if the user doesn’t already have the required version of Python (or any) installed. On Windows, and even on many Linux distributions and OS X, the right version of Python will not already be installed.

Besides, end-user software should always be in an executable format. Files ending in `.py` are for software engineers and system administrators.

One disadvantage of freezing is that it will increase the size of your distribution by about 2–12 MB. Also, you will be responsible for shipping updated versions of your application when security vulnerabilities to Python are patched.

Alternatives to Freezing

Packaging your code is for distributing libraries or tools to other developers.

On Linux, an alternative to freezing is to create a Linux distro package (e.g. .deb files for Debian or Ubuntu, or .rpm files for Red Hat and SuSE.)

Todo:

Fill in “Freezing Your Code” stub

Comparison of Freezing Tools

Solutions and platforms/features supported:

Solution	Windows	Linux	OS X	Python 3	License	One-file mode	Zipfile import	Eggs	pkg_resources support
bbFreeze	yes	yes	yes	no	MIT	no	yes	yes	yes
py2exe	yes	no	no	yes	MIT	yes	yes	no	no
pyInstaller	yes	yes	yes	yes	GPL	yes	no	yes	no
cx_Freeze	yes	yes	yes	yes	PSF	no	yes	yes	no
py2app	no	no	yes	yes	MIT	no	yes	yes	yes

Note:

Freezing Python code on Linux into a Windows executable was only once supported in PyInstaller and later dropped.

Note:

All solutions need a Microsoft Visual C++ to be installed on the target machine, except py2app. Only PyInstaller makes a self-executable exe that bundles the appropriate DLL when passing `--onefile` to `Configure.py`.

Windows

bbFreeze

Prerequisite is to install Python, Setuptools and pywin32 dependency on Windows.

1. Install `bbfreeze`:

   ```
   $ pip install bbfreeze
   ```

2. Write most basic `bb_setup.py`

   ```
   from bbfreeze import Freezer

   freezer = Freezer(distdir='dist')
   freezer.addScript('foobar.py', gui_only=True)
   freezer()
   ```

Note:

This will work for the most basic one file scripts. For more advanced freezing you will have to provide include and exclude paths like so:

```
freezer = Freezer(distdir='dist', includes=['my_code'], excludes=['docs'])
```

3. (Optionally) include icon

   ```
   freezer.setIcon('my_awesome_icon.ico')
   ```

4. Provide the Microsoft Visual C++ runtime DLL for the freezer. It might be possible to append your `sys.path` with

the Microsoft Visual Studio path but I find it easier to drop `msvcp90.dll` in the same folder where your script resides.

5. Freeze!

```
$ python bb_setup.py
```

py2exe

Prerequisite is to install Python on Windows. The last release of py2exe is from the year 2014. There is not active development.

1. Download and install http://sourceforge.net/projects/py2exe/files/py2exe/
2. Write `setup.py` (List of configuration options):

```
from distutils.core import setup
import py2exe

setup(
    windows=[{'script': 'foobar.py'}],
)
```

3. (Optionally) include icon
4. (Optionally) one-file mode
5. Generate `.exe` into `dist` directory:

```
$ python setup.py py2exe
```

6. Provide the Microsoft Visual C++ runtime DLL. Two options: globally install dll on target machine or distribute dll alongside with .exe.

PyInstaller

Prerequisite is to have installed Python, Setuptools and pywin32 dependency on Windows.

- Most basic tutorial
- Manual

OS X

py2app

PyInstaller

PyInstaller can be used to build Unix executables and windowed apps on Mac OS X 10.6 (Snow Leopard) or newer.

To install PyInstaller, use pip:

```
$ pip install pyinstaller
```

To create a standard Unix executable, from say `script.py`, use:

```
$ pyinstaller script.py
```

This creates:

- a `script.spec` file, analogous to a `make` file
- a `build` folder, that holds some log files
- a `dist` folder, that holds the main executable `script`, and some dependent Python libraries

all in the same folder as `script.py`. PyInstaller puts all the Python libraries used in `script.py` into the `dist` folder, so when distributing the executable, distribute the whole `dist` folder.

The `script.spec` file can be edited to customise the build, with options such as:

- bundling data files with the executable
- including run-time libraries (`.dll` or `.so` files) that PyInstaller can't infer automatically
- adding Python run-time options to the executable

Now `script.spec` can be run with `pyinstaller` (instead of using `script.py` again):

```
$ pyinstaller script.spec
```

To create a standalone windowed OS X application, use the `--windowed` option:

```
$ pyinstaller --windowed script.spec
```

This creates a `script.app` in the `dist` folder. Make sure to use GUI packages in your Python code, like PyQt or PySide, to control the graphical parts of the app.

There are several options in `script.spec` related to Mac OS X app bundles here. For example, to specify an icon for the app, use the `icon=\path\to\icon.icns` option.

Linux

bbFreeze

PyInstaller