- ← Index

Creating an Executable from a Python Script

Python is one of my favorite programming languages. That being said, if you've ever had to deploy an application written in Python then you know just how painful it can be.

Fortunately, there are some pretty awesome open-source tools that can be used to package a Python program into a standalone binary executable that contains everything needed to run the application (i.e. Python interpreter, program code, libraries, data, etc.).

In this article, I'll show you how to create a binary executable version of a graphical "Hello World" application using PyInstaller on Windows.

According to the PyInstaller website, PyInstaller supports all major operating systems, so if you're targeting a binary distribution on OS X or GNU/Linux systems, the process will likely be similar on those platforms.

First Things First

Of course, make sure that you already have Python 2.7.x installed.

The demo app in this article uses the wxPython library, so you will need to install that if you plan to follow along, but it is not necessary for using PyInstaller.

You will need to install PyInstaller as well, but I will get to that in a second.

The App

This app will be a simple "Hello World" graphical app. Save the source code below as **app.py**.

app.py

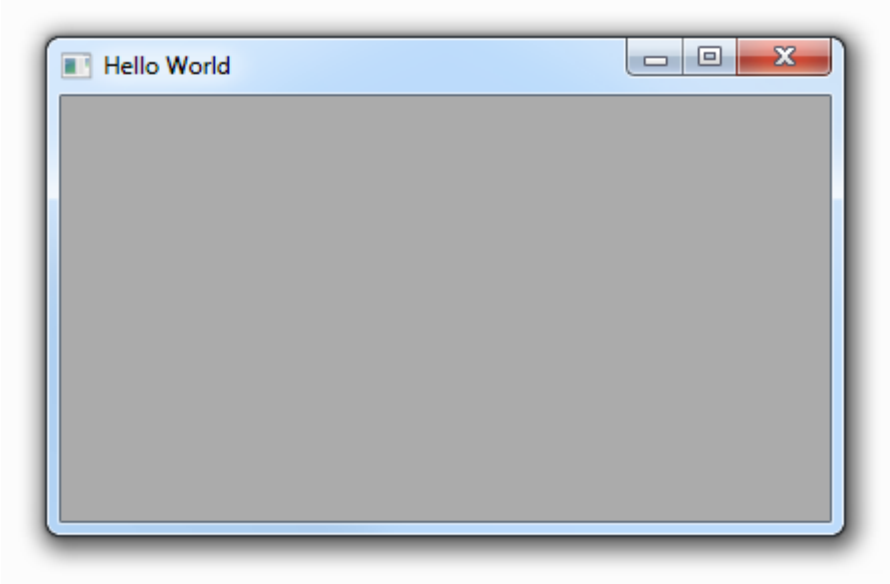
```
#!/usr/bin/env python
import wx
app = wx.App(False)
frame = wx.Frame(None, wx.ID_ANY, "Hello World")
frame.Show(True)
app.MainLoop()
```

Note: The source code shown above was taken from the wxPython Getting Started guide, which you may be interested in reading if you want to create graphical Python applications.

Now, fire up your console and run the app as usual.

```
python app.py
```

A single window appears as shown below.



It’s not very exciting, but this is just a demo.

Installing PyInstaller

Note: Before installing PyInstaller on Windows, you will need to install PyWin32. You do not need to do this for GNU/Linux or Mac OS X systems.

PyInstaller can be installed using Pip, the Python package manager.

```
pip install pyinstaller
```

Building the Executable

Now, build the executable.

```
pyinstaller.exe --onefile --windowed app.py
```

It’s that easy.

If the build was successful, the final executable, **app.exe**, and any associated files, will be placed in the **dist** directory, which will be created if it doesn’t exist.

Let me briefly describe the options that are being used:

- `--onefile` is used to package everything into a single executable. If you do not specify this option, the libraries, etc. will be distributed as separate files alongside the main executable.
- `--windowed` prevents a console window from being displayed when the application is run. If you're releasing a non-graphical application (i.e. a console application), you do not need to use this option.
- `app.py` the main source file of the application. The basename of this script will be used to name of the executable, however you may specify an alternative executable name using the `--name` option.

See the PyInstaller Manual for more configuration information.

You do not need to specify additional modules in the command as they will be automatically pulled via `import` statements.

Note: On my system the final executable is a sizable 8.4 MiB. The executable is relatively large because the Python interpreter, the application code, and all the required libraries are all packaged in (as specified by the `--onefile` option). Though convenient, there are some implications with this approach which you should be aware of before releasing using this method. See the PyInstaller Manual for more information on bundling.

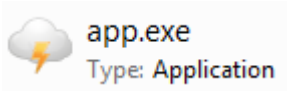
After the build, an **app.spec** file will be created. This file contains all of the options used to run PyInstaller, and can be fed back into PyInstaller for future builds in place of the command line options, if desired.

Adding an Icon

IconEden offers some nice royalty-free icons, so I'll use one of theirs for this demo. Save an .ico file in the source directory and add the `--icon=app.ico` option when you run PyInstaller. For example:

```
pyinstaller.exe --onefile --windowed --icon=app.ico app.py
```

This is an example of what an icon looks like when added to the application and viewed through Windows Explorer:



Adding Version Information

The following file (taken from the PyInstaller test suite) is used by PyInstaller to add version information to the executable. Save this file as **version.txt**.

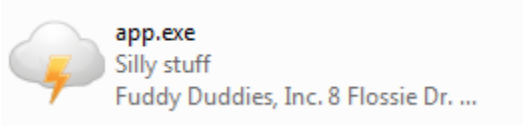
version.txt

```
# UTF-8
#
# For more details about fixed file info 'ffi' see:
# http://msdn.microsoft.com/en-us/library/ms646997.aspx
VSVersionInfo(
  ffi=FixedFileInfo(
    # filevers and prodvers should be always a tuple with four items: (1, 2,
3, 4)
    # Set not needed items to zero 0.
    filevers=(96, 12, 19, 1),
    prodvers=(4, 1, 2, 1),
    # Contains a bitmask that specifies the valid bits 'flags'
    mask=0x3f,
    # Contains a bitmask that specifies the Boolean attributes of the file.
    flags=0x0,
    # The operating system for which this file was designed.
    # 0x4 - NT and there is no need to change it.
    OS=0x4,
    # The general type of file.
    # 0x1 - the file is an application.
    fileType=0x1,
    # The function of the file.
    # 0x0 - the function is not defined for this fileType
    subtype=0x0,
    # Creation date and time stamp.
    date=(0, 0)
    ),
  kids=[
    StringFileInfo(
      [
      StringTable(
        u'040904b0',
        [StringStruct(u'CompanyName', u'Fuddy Duddies, Inc. 8 Flossie Dr.
Arlington, VA 00001'),
        StringStruct(u'ProductName', u'SILLINESS'),
        StringStruct(u'ProductVersion', u'2, 0, 3, 0'),
        StringStruct(u'InternalName', u'SILLINESS'),
        StringStruct(u'OriginalFilename', u'silliness.exe'),
        StringStruct(u'FileVersion', u'96, 12, 19, 1'),
        StringStruct(u'FileDescription', u'Silly stuff'),
        StringStruct(u'LegalCopyright', u'Copyright 2001 Fuddy Duddies,
Inc.'),
        StringStruct(u'LegalTrademarks', u'SILLINESS is a registered
trademark of Fuddy Duddies, Inc.'),])
      ]),
    VarFileInfo([VarStruct(u'Translation', [1033, 1200])])
  ]
)
```

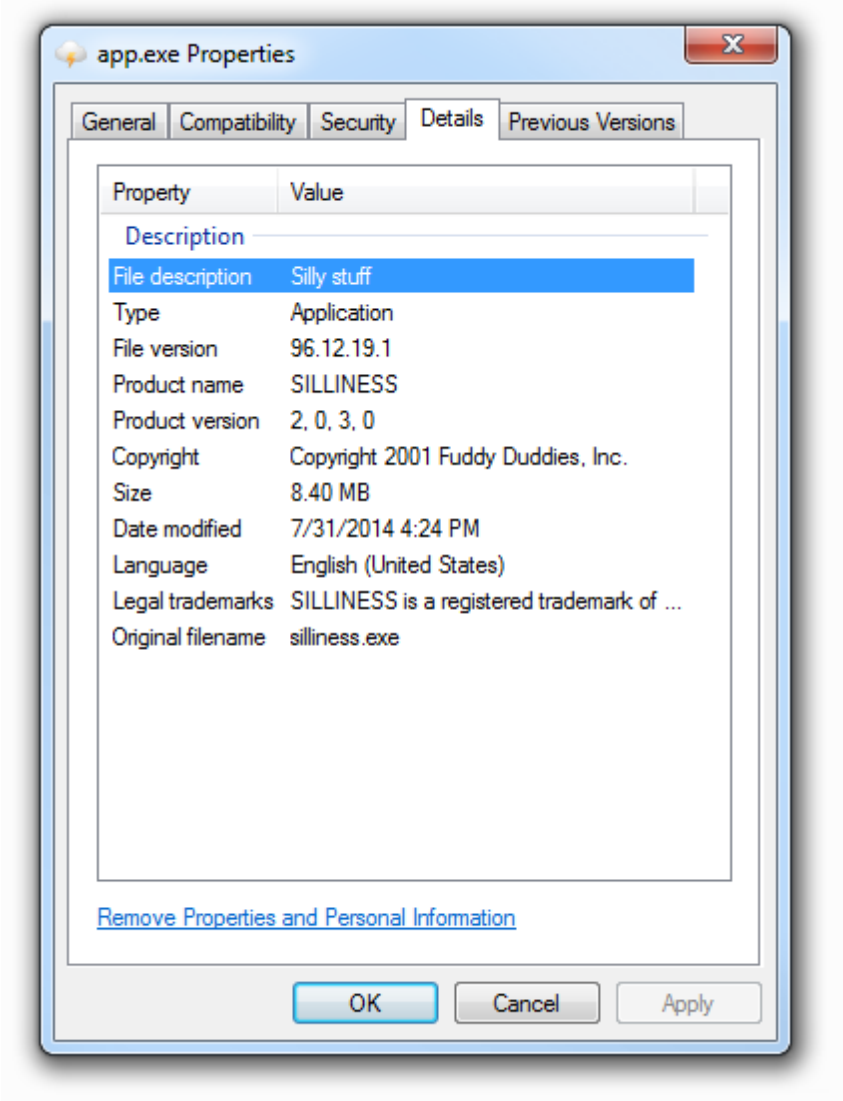
When you have your version.txt file ready, simply add the `--version-file=version.txt` option to the command line when you run PyInstaller. For example:

```
pyinstaller.exe --onefile --windowed --icon=app.ico --version-
file=version.txt app.py
```

Now, the application has an icon and version information.



Look at the Properties to see more information.



Final Thoughts

So that's about it. Pretty awesome, right?

Of course, I glossed over some of the complicated bits like runtime libraries, 3rd party modules, spec files, and program data files. The PyInstaller Manual covers all of this, so give it a read.

If PyInstaller isn't what you're looking for, here are some alternatives:

- Py2Exe
- Py2App (for Mac OS X)
- cx_Freeze

The next step for a proper deployment is to create an installer package (see Inno Setup and NSIS).

System Notes

- Windows 7 SP1 (64-bit)
- Python 2.7.6 (32-bit)
- PyWin32 219
- PyInstaller 2.1
- wxPython 3.0

- Written by Matt Borgerson
- Published on July 30, 2014

 Powered by coffee.