cx_Freeze Documentation

Release 5.0.2

Anthony Tuininga

May 20, 2017

Contents

cx_Freeze is a set of scripts and modules for freezing Python scripts into executables in much the same way that py2exe and py2app do. Unlike these two tools, cx_Freeze is cross platform and should work on any platform that Python itself works on. It requires Python 2.7 or higher and does work with Python 3.

cx_Freeze is distributed under an open-source *license* (the PSF license).

Contents:

CHAPTER 1

Using cx_Freeze

There are three different ways to use cx_Freeze:

1. Use the included *cxfreeze script*.
2. Create a *distutils setup script*. This is useful if you need extra options when freezing your program, because you can save them in the script. Run `cxfreeze-quickstart` to generate a simple setup script.
3. Work directly with the classes and modules used internally by cx_Freeze. This should be reserved for complicated scripts or extending or embedding.

cx_Freeze normally produces a folder containing an executable file for your program, along with the shared libraries (DLLs or .so files) needed to run it. You can make a simple Windows installer using a *setup script* with the `bdist_msi` option, or a Mac disk image with `bdist_dmg`. For a more advanced Windows installer, use a separate tool like Inno Setup to package the files cx_Freeze collects.

Python modules for your executables are stored in a zip file. Packages are stored in the file system by default but can also be included in the zip file.

CHAPTER 2

distutils setup script

In order to make use of distutils a setup script must be created. This is called `setup.py` by convention, although it can have any name. It looks something like this:

```
import sys
from cx_Freeze import setup, Executable

# Dependencies are automatically detected, but it might need fine tuning.
build_exe_options = {"packages": ["os"], "excludes": ["tkinter"]}

# GUI applications require a different base on Windows (the default is for a
# console application).
base = None
if sys.platform == "win32":
    base = "Win32GUI"

setup(  name = "guifoo",
        version = "0.1",
        description = "My GUI application!",
        options = {"build_exe": build_exe_options},
        executables = [Executable("guifoo.py", base=base)])
```

There are more examples in the `samples/` directory of the source.

The script is invoked as follows:

```
python setup.py build
```

This command will create a subdirectory called `build` with a further subdirectory starting with the letters `exe.` and ending with the typical identifier for the platform that distutils uses. This allows for multiple platforms to be built without conflicts.

On Windows, you can build a simple installer containing all the files cx_Freeze includes for your application, by running the setup script as:

```
python setup.py bdist_msi
```

On Mac OS X, you can use `bdist_dmg` to build a Mac disk image.

distutils commands

cx_Freeze creates four new commands and subclasses four others in order to provide the ability to both build and install executables. In typical distutils fashion they can be provided in the setup script, on the command line or in a `setup.cfg` configuration file. They are described in further detail below.

To specify options in the script, use underscores in the name. For example:

```
setup(...
      options = {'build_exe': {'init_script':'Console'}} )
```

To specify the same options on the command line, use dashes, like this:

```
python setup.py build_exe --init-script Console
```

Some options also have a short form to use on the command line. These are given in brackets below.

build

This command is a standard command which has been modified by cx_Freeze to build any executables that are defined. The following options were added to the standard set of options for the command:

option name	description
build_exe (-b)	directory for built executables and dependent files, defaults to `build/`

build_exe

This command performs the work of building an executable or set of executables. It can be further customized:

option name	description
build_exe (-b)	directory for built executables and dependent files, defaults to `build/`
optimize (-o)	optimization level, one of 0 (disabled), 1 or 2
excludes (-e)	comma separated list of names of modules to exclude
includes (-e)	comma separated list of names of modules to include
packages (-p)	comma separated list of packages to include, which includes all submodules in the package
namespace_packages	comma separated list of packages to be treated as namespace packages (path is extended using pkgutil)
replace_paths	Modify filenames attached to code objects, which appear in tracebacks. Pass a comma separated list of paths in the form <search>=<replace>. The value * in the search portion will match the directory containing the entire package, leaving just the relative path to the module.
path	comma separated list of paths to search; the default value is sys.path
compressed (-c)	create a compressed zip file
constants	comma separated list of constant values to include in the constants module called BUILD_CONSTANTS in form <name>=<value>
include_files	list containing files to be copied to the target directory; it is expected that this list will contain strings or 2-tuples for the source and destination; the source can be a file or a directory (in which case the tree is copied except for .svn and CVS directories); the target must not be an absolute path
include_msvcr	include the Microsoft Visual C runtime DLLs and (if necessary) the manifest file required to run the executable without needing the redistributable package installed
zip_includes	list containing files to be included in the zip file directory; it is expected that this list will contain strings or 2-tuples for the source and destination
bin_includes	list of names of files to include when determining dependencies of binary files that would normally be excluded; note that version numbers that normally follow the shared object extension are stripped prior to performing the comparison
bin_excludes	list of names of files to exclude when determining dependencies of binary files that would normally be included; note that version numbers that normally follow the shared object extension are stripped prior to performing the comparison
bin_path_includes	list of paths from which to include files when determining dependencies of binary files
bin_path_excludes	list of paths from which to exclude files when determining dependencies of binary files
zip_include_packages	list of packages which should be included in the zip file; the default is for all packages to be placed in the file system, not the zip file; those packages which are known to work well inside a zip file can be included if desired; use * to specify that all packages should be included in the zip file
zip_exclude_packages	list of packages which should be excluded from the zip file and placed in the file system instead; the default is for all packages to be placed in the file system since a number of packages assume that is where they are found and will fail when placed in a zip file; use * to specify that all packages should be placed in the file system and excluded from the zip file (the default)
silent (-s)	suppress all output except warnings

install

This command is a standard command which has been modified by cx_Freeze to install any executables that are defined. The following options were added to the standard set of options for the command:

option name	description
install_exe	directory for installed executables and dependent files

install_exe

This command performs the work installing an executable or set of executables. It can be used directly but most often is used when building Windows installers or RPM packages. It can be further customized:

option name	description
install_dir (-d)	directory to install executables to; this defaults to a subdirectory called <name>-<version> in the "Program Files" directory on Windows and <prefix>/lib on other platforms; on platforms other than Windows symbolic links are also created in <prefix>/bin for each executable.
build_dir (-b)	build directory (where to install from); this defaults to the build_dir from the build command
force (-f)	force installation, overwriting existing files
skip_build	skip the build steps

bdist_msi

This command is a standard command in Python 2.5 and higher which has been modified by cx_Freeze to handle installing executables and their dependencies. The following options were added to the standard set of options for the command:

option name	description
add_to_path	add the target directory to the PATH environment variable; the default value is True if there are any console based executables and False otherwise
upgrade_code	define the upgrade code for the package that is created; this is used to force removal of any packages created with the same upgrade code prior to the installation of this one

bdist_rpm

This command is a standard command which has been modified by cx_Freeze to ensure that packages are created with the proper architecture for the platform. The standard command assumes that the package should be architecture independent if it cannot find any extension modules.

bdist_mac

This command is available on Mac OS X systems, to create a Mac application bundle (a .app directory).

option name	description
iconfile	Path to an icns icon file for the application. This will be copied into the bundle.
qt_menu_nib	Path to the qt-menu.nib file for Qt applications. By default, it will be auto-detected.
bundle_name	File name for the bundle application without the .app extension.
custom_info_plist	File to be used as the Info.plist in the app bundle. A basic one will be generated by default.
include_frameworks	A list of Framework directories to include in the app bundle.
codesign_identity	The identity of the key to be used to sign the app bundle.
codesign_entitlements	The path to an entitlements file to use for your application's code signature.
codesign_deep	Boolean for whether to codesign using the –deep option.
codesign_resource_rules	Plist file to be passed to codesign's –resource-rules option.

New in version 4.3.

Changed in version 4.3.2: Added the `iconfile` and `bundle_name` options.

Changed in version 4.3.3: Added the `include_frameworks`, `custom_info_plist`, `codesign_identity` and `codesign_entitlements` options.

Changed in version 4.3.4: Added the `codesign_deep` and `codesign_resource_rules` options.

bdist_dmg

This command is available on Mac OS X systems; it creates an application bundle, then packages it into a DMG disk image suitable for distribution and installation.

option name	description
volume_label	Volume label of the DMG disk image
applications_shortcut	Boolean for whether to include shortcut to Applications in the DMG disk image

New in version 4.3.

Changed in version 4.3.2: Added the `applications_shortcut` option.

cx_Freeze.Executable

The options for the *build_exe* command are the defaults for any executables that are created. The options for the *Executable* class allow specification of the values specific to a particular executable. The arguments to the constructor are as follows:

argument name	description
script	the name of the file containing the script which is to be frozen
initScript	the name of the initialization script that will be executed before the actual script is executed; this script is used to set up the environment for the executable; if a name is given without an absolute path the names of files in the initscripts subdirectory of the cx_Freeze package is searched
base	the name of the base executable; if a name is given without an absolute path the names of files in the bases subdirectory of the cx_Freeze package is searched
targetName	the name of the target executable; the default value is the name of the script with the extension exchanged with the extension for the base executable
icon	name of icon which should be included in the executable itself on Windows or placed in the target directory for other platforms
shortcutName	the name to give a shortcut for the executable when included in an MSI package (Windows only).
shortcutDir	the directory in which to place the shortcut when being installed by an MSI package; see the MSI Shortcut table documentation for more information on what values can be placed here (Windows only).
copyright	the copyright value to include in the version resource associated with executable (Windows only).
trademarks	the trademarks value to include in the version resource associated with executable (Windows only).

CHAPTER 3

cxfreeze script

The `cxfreeze` script is included with other Python scripts. On Windows and the Mac this is in the `Scripts` subdirectory of your Python installation whereas on Unix platforms this in the bin directory of the prefix where Python is installed.

Assuming you have a script called `hello.py` which you want to turn into an executable, this can be accomplished by this command:

```
cxfreeze hello.py --target-dir dist
```

Further customization can be done using the following options:

--version
: show version number and exit

-h, --help
: show this help message and exit

-O
: optimize generated bytecode as per PYTHONOPTIMIZE; use -OO in order to remove doc strings

-c, --compress
: compress byte code in zip files

-s, --silent
: suppress all output except warnings and errors

--base-name=NAME
: file on which to base the target file; if the name of the file is not an absolute file name, the subdirectory bases (rooted in the directory in which the freezer is found) will be searched for a file matching the name

--init-script=NAME
: script which will be executed upon startup; if the name of the file is not an absolute file name, the subdirectory initscripts (rooted in the directory in which the cx_Freeze package is found) will be searched for a file matching the name

--target-dir=DIR, **--install-dir**=DIR
: The directory in which to place the target file and any dependent files

--target-name=NAME
: the name of the file to create instead of the base name of the script and the extension of the base binary

--default-path=DIRS
: list of paths separated by the standard path separator for the platform which will be used to initialize sys.path prior to running the module finder

--include-path=DIRS
: list of paths separated by the standard path separator for the platform which will be used to modify sys.path prior to running the module finder

--replace-paths=DIRECTIVES
: replace all the paths in modules found in the given paths with the given replacement string; multiple values are separated by the standard path separator and each value is of the form path=replacement_string; path can be * which means all paths not already specified

--include-modules=NAMES
: comma separated list of modules to include

--exclude-modules=NAMES
: comma separated list of modules to exclude

--ext-list-file=NAME
: name of file in which to place the list of dependent files which were copied into the target directory

-z SPEC, **--zip-include**=SPEC
: name of file to add to the zip file or a specification of the form name=arcname which will specify the archive name to use; multiple –zip-include arguments can be used

--icon=ICON
: name of the icon file for the application

CHAPTER 4

Frequently Asked Questions

Problems with running frozen programs

A common problem is that cx_Freeze hasn't automatically detected that a file needs to be copied. Modules that your code imports are detected, but if they're dynamically loaded - e.g. by a plugin system - you have to tell cx_Freeze about them. This is easy using a *setup script*:

- For Python code, specify the module names in the `includes` or `packages` options.
- List compiled libraries (.dll or .so files) in the `include_files` option.
- Data files are a bit more complex - see *Using data files*.

Windows command prompt appears briefly

If there's a problem with your frozen application, you may see a command prompt window appear briefly when you try to run it, and then disappear again. This happens when a console-mode executable exits quickly, usually if there's an error as soon as it starts.

There are two ways to debug what's going on:

1. Freeze your application with the `Win32GUI` base (see *distutils setup script* or *cxfreeze script*). This doesn't use a console window, and reports errors in a dialog box.
2. Alternatively, start a command prompt yourself and launch the frozen executable from the command line. This will let you see any error messages in the console.

Freezing for other platforms

cx_Freeze works on Windows, Mac and Linux, but on each platform it only makes an executable that runs on that platform. So if you want to freeze your program for Windows, freeze it on Windows; if you want to run it on Macs, freeze it on a Mac.

At a pinch, you can try to make a Windows executable using Wine. Our experience is that you need to copy some files in manually after cx_Freeze has run to make the executable work. We don't recommend this option.

Using data files

Applications often need data files besides the code, such as icons. Using a *setup script*, you can list data files or directories in the `include_files` option to `build_exe`. They'll be copied to the build directory alongside the executable. Then to find them, use code like this:

```
def find_data_file(filename):
    if getattr(sys, 'frozen', False):
        # The application is frozen
        datadir = os.path.dirname(sys.executable)
    else:
        # The application is not frozen
        # Change this bit to match where you store your data files:
        datadir = os.path.dirname(__file__)

    return os.path.join(datadir, filename)
```

An alternative is to embed data in code, for example by using Qt's resource system.

See also:

A tutorial covering resources in PyQt

Microsoft Visual C++ Redistributable Package

Python on Windows requires the Microsoft Visual C++ Redistributable Package. Python 2.6-3.2 uses the 2008 version, and because of how this is installed, cx_Freeze doesn't automatically copy it for your application. It's also not clear whether everyone has the right to redistribute the DLLs. You're responsible for checking the license conditions associated with the DLLs you have installed.

- If your license allows you to distribute these files, specify the `include_msvcr` option to *build_exe* to have them distributed automatically.
- If not, your users or your installer will need to install the Microsoft Visual C++ Redistributable Package (a free download from Microsoft). It's not uncommon for this to already be present on modern computers, but it's not, as far as we know, part of a standard Windows installation. Note that the "SP1" version of this *does not* work – it has to exactly match the version which Python itself is compiled with.
  - 2008 (Python 2.6-3.2) for x86 (32 bit) Windows or for x64 (64 bit) Windows
  - 2010 (Python 3.3) for x86 (32 bit) Windows or for x64 (64 bit) Windows

Up to Python 2.5, and again from Python 3.3, the MSVCR DLLs are installed in a normal location, and cx_Freeze will copy them automatically. It's still up to you to ensure that the licenses of all the files you use allow you to distribute them as part of your application.

Single-file executables

cx_Freeze does not support building a single file exe, where all of the libraries for your application are embedded in one executable file.

You can use IExpress to compress the build directory from cx_Freeze into a self-extracting archive: an exe which unpacks your application into a temporary directory and runs it. IExpress is a utility that's included with Windows, intended for making installers, but it works equally well if you tell it to run the cx_Freeze-built exe after extraction.

Alternatively, you can create a self extracting archive using 7zip. This is a bit more complex than using IExpress, but might provide more flexibility, and allows you to build your application using only open source tools.

CHAPTER 5

Release notes

5.x releases

Development version

Note: This version supports Python 2.7 and above.

Version 5.0.2 (May 2017)

1. Correct handling of import in child thread (PR #245)
2. Correct handling of "dis" module with Python 3.5.1 (Issue #225)
3. Correct handling of "multiprocess.process" module (Issue #230)
4. Correct attempt to assign variable to an empty list (PR #260)
5. Improved README (PR #235, PR #236)
6. Add hook for pythonnet package (PR #251)
7. Add hook for sqlite3 and improve win32file hook (PR #261)
8. Add FAQ entry (PR #267)

Version 5.0.1 (January 2017)

1. Added support for Python 3.6.
2. Corrected hooks for the pythoncom and pywintypes modules.
3. Use realpath() to get the absolute path of the executable; this resolves symbolic links and ensures that changing the path before all imports are complete does not result in the executable being unable to find modules.

4. Correct issue with usage of 'if __main__ == "__main__"'. (Issue #211)
5. Correct handling of the zip_include_packages option. (Issue #208)
6. Correct logic regarding importing of submodules. (Issue #219)

Version 5.0 (November 2016)

Note: This version supports Python 2.7 and above.

1. Added support for Python 3.5.
2. Switched from using C compiled frozen modules which embed part of the standard library to using the default named zip file and library file locations. This eliminates the need to recompile cx_Freeze for each new Python version as no parts of the standard library are included in the installation now. This also implies that appending a zip file to the executable is no longer supported since the standard name and location are used.
3. Removed unnecessary options and parameters from cx_Freeze. (PR #60, PR #67)
4. Added support for Win32Service base with Python 3.x. (PR #49)
5. Add __version__ as an alias to version. (PR #65)
6. Updated hooks for PyQt, h5py. (PR #68, PR #64, PR #70)
7. Set copyDependentFiles = True for include files. (PR #66)
8. Reallow including modules with non-identifier names. (PR #79)
9. Fix missing space in Windows installer. (PR #81)
10. Use pattern "not in string" isntead of "string.find(pattern)" (PR #76)
11. Fix –add-to-path writing to the per-user instead of system environment (PR #86)
12. Fix documentation (PR #77, PR #78)
13. Do not import excluded submodules. (PR #89)
14. Correct distribution files for bdist_msi (PR #95)
15. Allow proper handling of Unicode command line parameters under Windows (PR #87)
16. Add pyzmq hook (PR #63)
17. Add copyright and trademarks to version information (PR #94)
18. Fix compilation on Ubuntu (Issue #32)
19. Set defaults in class directly, rather than as defaults in the function signature. (Issue #185)
20. Correct relative import of builtin module (cx_Freeze was incorrectly considering it an extension found within a package). (Issue #127)
21. Ensure that included files are added relative to the executable, not to the location of the zip file. (Issue #183)
22. Prevent infinite loop while using cx_Freeze installed in a prefix. (Issue #204)
23. Added support for storing packages in the file system instead of in the zip file. There are a number of packages that assume that they are found in the file system and if found in a zip file instead produce strange errors. The default is now to store packages in the file system but a method is available to place packages in the zip file if they are known to behave properly when placed there. (Issue #73)

24. Added support for untranslatable characters on Windows in the path where a frozen executable is located. (Issue #29)
25. Use volume label to name the DMG file (Issue #97)
26. Significantly simplified startup code.
27. Added logging statements for improved debugging.
28. Updated samples to handle recent updates to packages.
29. Avoid infinite loop for deferred imports which are cycles of one another.

Version 4.3.4 (December 2014)

Note: This version supports Python 2.6 and above.

1. Rebuilt for Python 3.4.2. Dropped support for Python versions less than 2.6.
2. Correct stale comment. (PR #50)
3. Fix processing path specs from config when targets are not explicit. (PR #53)
4. Tweaks to improve compiling with MSVC 10 (2010) on Windows. (PR #54)
5. Added support for using the –deep and –resource-rules options when code signing through cx_Freeze on OS X. (PR #55)
6. Catch error if GetDependentFiles() is called on a non-library (PR #56)
7. Added FAQ entry on single file executables (PR #58)
8. Only look one level deep for implicit relative imports (PR #59)
9. Removed statement that was filtering out the ntpath module. (PR #74)

Version 4.3.3 (May 2014)

Note: This version supports Python 2.4 and above.

1. Added support for release version of 3.4 (PR #47, PR #48)
2. Added support for code signing in bdist_mac (PR #40)
3. Added custom Info.plist and Framework suport to bdist_mac (PR #33)
4. Added support for resolving dependencies on OS X where paths are relative (PR #35)
5. Added hook for QtWebKit module (PR #36)
6. Added support for finding packages inside zip files (PR #38)
7. Ensure that syntax errors in code do not prevent freezing from taking place but simply ignore those modules (PR #44, PR #45)
8. Init scripts now use code that works in both Python 2 and 3 (PR #42)
9. Simplify service sample (PR #41)
10. Fix documentation for bdist_dmg (PR #34)

11. All options that accept multiple values are split on commas as documented (PR #39)
12. Truncated names in Python tracebacks (Issue #52)
13. install_name_tool doesn't set relative paths for files added using include_files option (Issue #31)

Version 4.3.2 (October 2013)

1. Added support for Python 3.4.
2. Added hooks for PyQt4, PyQt5 and PySide to handle their plugins.
3. Added support for creating a shortcut/alias to the Applications directory within distributed DMG files for OS X.
4. Improve missing modules output.
5. Avoid polluting the extension module namespace when using the bootstrap module to load the extension.
6. Added support for using setuptools and pip if such tools are available.
7. Added first tests; nose and mock are required to run them.
8. Remove –bundle-iconfile in favor of –iconfile as a more generic method of including the icon for bdist_mac.
9. Documentation improved and FAQ added.
10. Converted samples to follow PEP 8.
11. cxfreeze-quickstart failed if the default base was not used
12. scripts frozen with Python 3 fail with an ImportError trying to import the re module
13. fix bug where after a first attempt to find a module failed, the second attempt would erroneously succeed
14. stop attempting to load a module by a name that is not a valid Python identifier

Version 4.3.1 (November 2012)

Note: This version supports Python 2.4 and above. If you need Python 2.3 support, please use cx_Freeze 4.2.3.

1. Added support for the final release of Python 3.3.
2. Added support for copying the MSVC runtime DLLs and manifest if desired by using the –include-msvcr switch. Thanks to Almar Klein for the initial patch.
3. Clarified the documentation on the –replace-paths option. Thanks to Thomas Kluyver for the patch.
4. Producing a Mac distribution failed with a variable reference.
5. Freezing a script using PyQt on a Mac failed with a type error.
6. Version number reported was incorrect. (Issue #7)
7. Correct paths during installation on Windows. (Issue #11)

Version 4.3 (July 2012)

Note: This version supports Python 2.4 and above. If you need Python 2.3 support, please use cx_Freeze 4.2.3.

1. Added options to build Mac OS X application bundles and DMG packages using `bdist_mac` and `bdist_dmg` distutils commands. Written by Rob Reilink.
2. The documentation is now using Sphinx, and is available on ReadTheDocs.org.
3. Added support for Python 3.3 which uses a different compiled file format than earlier versions of Python.
4. Added support for Windows services which start automatically and which are capable of monitoring changes in sessions such as lock and unlock.
5. New `cxfreeze-quickstart` wizard to create a basic `setup.py` file. Initially written by Thomas Kluyver.
6. Included files under their original name can now be passed to `include_files` as a tuple with an empty second element. Written by r_haritonov.
7. File inclusions/exclusions can now be specified using a full path, or a shared library name with a version number suffix.
8. Messagebox display of certain errors in Windows GUI applications with Python 3.
9. Running Windows GUI applications in a path containing non-ASCII characters.
10. Calculate the correct filename for the Python shared library in Python 3.2.
11. Including a package would not include the packages within that package, only the modules within that package. (Issue #3)

Version 4.2.3 (March 2011)

1. Added support for Python 3.2.
2. Added hook for datetime module which implicitly imports the time module.
3. Fixed hook for tkinter in Python 3.x.
4. Always include the zlib module since the zipimport module requires it, even when compression is not taking place.
5. Added sample for a tkinter application.

Version 4.2.2 (December 2010)

1. Added support for namespace packages which are loaded implicitly upon startup by injection into sys.modules.
2. Added support for a Zope sample which makes use of namespace packages.
3. Use the Microsoft compiler on Windows for Python 2.6 and up as some strange behaviors were identified with Python 2.7 when compiled using mingw32.
4. Eliminate warning about -mwindows when using the Microsoft compiler for building the Win32GUI base executable.
5. Added support for creating version resources on Windows.
6. Ensure that modules that are not truly required for bootstrapping are not included in the frozen modules compiled in to the executable; otherwise, some packages and modules (such as the logging package) cannot be found at runtime. This problem only seems to be present in Python 2.7.1 but it is a good improvement for earlier releases of Python as well.
7. Added support for setting the description for Windows services.
8. Added hook for using the widget plugins which are part of the PyQt4.uic package.

9. Added additional hooks to remove spurious errors about missing modules and to force inclusion of implicitly imported modules (twitter module and additional submodules of the PyQt4 package).
10. Fixed support for installing frozen executables under Python 3.x on Windows.
11. Removed optional import of setuptools which is not a complete drop-in replacement for distutils and if found, replaces distutils with itself, resulting in some distutils features not being available; for those who require or prefer the use of setuptools, import it in your setup.py.

Version 4.2.1 (October 2010)

1. Added support for specifying bin_path_includes and bin_path_excludes in setup scripts.
2. Added support for compiling Windows services with the Microsoft compiler and building for 64-bit Windows.
3. When installing Windows services, use the full path for both the executable and the configuration file if specified.
4. Eliminate duplicate files for each possible version of Python when building MSI packages for Python 2.7.
5. Fix declaration of namespace packages.
6. Fix check for cx_Logging import library directory.
7. Added hooks for the python-Xlib package.
8. Added hooks to ignore the _scproxy module when not on the Mac platform and the win32gui and pyHook modules on platforms other than Windows.
9. When copying files, copy the stat() information as well as was done in earlier versions of cx_Freeze.
10. Added documentation for the shortcutName and shortcutDir parameters for creating an executable.

Version 4.2 (July 2010)

1. Added support for Python 2.7.
2. Improved support for Python 3.x.
3. Improved support for Mac OS X based on feedback from some Mac users.
4. Improved hooks for the following modules: postgresql, matplotlib, twisted, zope, PyQt4.
5. Improved packaging of MSI files by enabling support for creating shortcuts for the executables, for specifying the initial target directory and for adding other arbitrary configuration to the MSI.
6. Added support for namespace packages such as those distributed for zope.
7. The name of the generated MSI packages now includes the architecture in order to differentiate between 32-bit and 64-bit builds.
8. Removed use of LINKFORSHARED on the Mac which is not necessary and for Python 2.6 at least causes an error to be raised.
9. Turn off filename globbing on Windows as requested by Craig McQueen.
10. Fixed bug that prevented hooks from successfully including files in the build (as is done for the matplotlib sample).
11. Fixed bug that prevented submodules from being included in the build if the format of the import statement was from . import <name>.

12. Reverted bug fix for threading shutdown support which has been fixed differently and is no longer required in Python 2.6.5 and up (in fact an error is raised if the threading module is used in a frozen executable and this code is retained).
13. Fixed bug which resulted in attempts to compile .pyc and .pyo files from the initscripts directory.
14. Fixed selection of "Program Files" directory on Windows in 64-bit MSI packages built by cx_Freeze.

Version 4.1.2 (January 2010)

1. Fix bug that caused the util extension to be named improperly.
2. Fix bug that prevented freezing from taking place if a packaged submodule was missing.
3. Fix bug that prevented freezing from taking place in Python 3.x if the encoding of the source file wasn't compatible with the encoding of the terminal performing the freeze.
4. Fix bug that caused the base modules to be included in the library.zip as well as the base executables.

Version 4.1.1 (December 2009)

1. Added support for Python 3.1.
2. Added support for 64-bit Windows.
3. Ensured that setlocale() is called prior to manipulating file names so that names that are not encoded in ASCII can still be used.
4. Fixed bug that caused the Python shared library to be ignored and the static library to be required or a symbolic link to the shared library created manually.
5. Added support for renaming attributes upon import and other less frequently used idioms in order to avoid as much as possible spurious errors about modules not being found.
6. Force inclusion of the traceback module in order to ensure that errors are reported in a reasonable fashion.
7. Improved support for the execution of ldd on the Solaris platform as suggested by Eric Brunel.
8. Added sample for the PyQt4 package and improved hooks for that package.
9. Enhanced hooks further in order to perform hidden imports and avoid errors about missing modules for several additional commonly used packages and modules.
10. Readded support for the zip include option.
11. Avoid the error about digest mismatch when installing RPMs by modifying the spec files built with cx_Freeze.
12. Ensure that manifest.txt is included in the source distribution.

Version 4.1 (July 2009)

1. Added support for Python 3.x.
2. Added support for services on Windows.
3. Added command line option –silent (-s) as requested by Todd Templeton. This option turns off all normal output including the report of the modules that are included.
4. Added command line option –icon as requested by Tom Brown.

5. Ensure that Py_Finalize() is called even when exceptions take place so that any finalization (such as __del__ calls) are made prior to the executable terminating.
6. Ensured that empty directories are created as needed in the target as requested by Clemens Hermann.
7. The encodings package and any other modules required to bootstrap the Python runtime are now automatically included in the frozen executable.
8. Ensured that if a target name is specified, that the module name in the zip file is also changed. Thanks to Clemens Hermann for the initial patch.
9. Enabled support for compiling on 64-bit Windows.
10. If an import error occurs during the load phase, treat that as a bad module as well. Thanks to Tony Meyer for pointing this out.
11. As suggested by Todd Templeton, ensured that the include files list is copied, not simply referenced so that further uses of the list do not inadvertently cause side effects.
12. As suggested by Todd Templeton, zip files are now closed properly in order to avoid potential corruption.
13. As suggested by Todd Templeton, data files are no longer copied when the copy dependent files flag is cleared.
14. Enabled better support of setup.py scripts that call other setup.py scripts such as the ones used by cx_OracleTools and cx_OracleDBATools.
15. On Solaris, ldd outputs tabs instead of spaces so expand them first before looking for the separator. Thanks to Eric Brunel for reporting this and providing the solution.
16. On Windows, exclude the Windows directory and the side-by-side installation directory when determining DLLs to copy since these are generally considered part of the system.
17. On Windows, use %* rather than the separated arguments in the generated batch file in order to avoid problems with the very limited argument processor used by the command processor.
18. For the Win32GUI base executable, add support for specifying the caption to use when displaying error messages.
19. For the Win32GUI base executable, add support for calling the excepthook for top level exceptions if one has been specified.
20. On Windows, ensure that the MSI packages that are built are per-machine by default as otherwise strange things can happen.
21. Fixed bug in the calling of readlink() that would occasionally result in strange behavior or segmentation faults.
22. Duplicate warnings about libraries not found by ldd are now suppressed.
23. Tweaked hooks for a number of modules based on feedback from others or personal experience.

Version 4.0.1 (October 2008)

1. Added support for Python 2.6. On Windows a manifest file is now required because of the switch to using the new Microsoft C runtime.
2. Ensure that hooks are run for builtin modules.

Version 4.0 (September 2008)

1. Added support for copying files to the target directory.
2. Added support for a hook that runs when a module is missing.

3. Added support for binary path includes as well as excludes; use sequences rather than dictionaries as a more convenient API; exclude the standard locations for 32-bit and 64-bit libaries in multi-architecture systems.
4. Added support for searching zip files (egg files) for modules.
5. Added support for handling system exit exceptions similarly to what Python does itself as requested by Sylvain.
6. Added code to wait for threads to shut down like the normal Python interpreter does. Thanks to Mariano Disanzo for discovering this discrepancy.
7. Hooks added or modified based on feedback from many people.
8. Don't include the version name in the display name of the MSI.
9. Use the OS dependent path normalization routines rather than simply use the lowercase value as on Unix case is important; thanks to Artie Eoff for pointing this out.
10. Include a version attribute in the cx_Freeze package and display it in the output for the –version option to the script.
11. Include build instructions as requested by Norbert Sebok.
12. Add support for copying files when modules are included which require data files to operate properly; add support for copying the necessary files for the Tkinter and matplotlib modules.
13. Handle deferred imports recursively as needed; ensure that from lists do not automatically indicate that they are part of the module or the deferred import processing doesn't actually work!
14. Handle the situation where a module imports everything from a package and the __all__ variable has been defined but the package has not actually imported everything in the __all__ variable during initialization.
15. Modified license text to more closely match the Python Software Foundation license as was intended.
16. Added sample script for freezing an application using matplotlib.
17. Renamed freeze to cxfreeze to avoid conflict with another package that uses that executable as requested by Siegfried Gevatter.

Version 4.0b1 (September 2007)

1. Added support for placing modules in library.zip or in a separate zip file for each executable that is produced.
2. Added support for copying binary dependent files (DLLs and shared libraries)
3. Added support for including all submodules in a package
4. Added support for including icons in Windows executables
5. Added support for constants module which can be used for determining certain build constants at runtime
6. Added support for relative imports available in Python 2.5 and up
7. Added support for building Windows installers (Python 2.5 and up) and RPM packages
8. Added support for distutils configuration scripts
9. Added support for hooks which can force inclusion or exclusion of modules when certain modules are included
10. Added documentation and samples
11. Added setup.py for building the cx_Freeze package instead of a script used to build only the frozen bases
12. FreezePython renamed to a script called freeze in the Python distribution
13. On Linux and other platforms that support it set LD_RUN_PATH to include the directory in which the executable is located

Older versions

Version 3.0.3 (July 2006)

1. In Common.c, used MAXPATHLEN defined in the Python OS independent include file rather than the PATH_MAX define which is OS dependent and is not available on IRIX as noted by Andrew Jones.
2. In the initscript ConsoleSetLibPath.py, added lines from initscript Console.py that should have been there since the only difference between that script and this one is the automatic re-execution of the executable.
3. Added an explicit "import encodings" to the initscripts in order to handle Unicode encodings a little better. Thanks to Ralf Schmitt for pointing out the problem and its solution.
4. Generated a meaningful name for the extension loader script so that it is clear which particular extension module is being loaded when an exception is being raised.
5. In MakeFrozenBases.py, use distutils to figure out a few more platform-dependent linker flags as suggested by Ralf Schmitt.

Version 3.0.2 (December 2005)

1. Add support for compressing the byte code in the zip files that are produced.
2. Add better support for the win32com package as requested by Barry Scott.
3. Prevent deletion of target file if it happens to be identical to the source file.
4. Include additional flags for local modifications to a Python build as suggested by Benjamin Rutt.
5. Expanded instructions for building cx_Freeze from source based on a suggestion from Gregg Lind.
6. Fix typo in help string.

Version 3.0.1 (December 2004)

1. Added option –default-path which is used to specify the path used when finding modules. This is particularly useful when performing cross compilations (such as for building a frozen executable for Windows CE).
2. Added option –shared-lib-name which can be used to specify the name of the shared library (DLL) implementing the Python runtime that is required for the frozen executable to work. This option is also particularly useful when cross compiling since the normal method for determining this information cannot be used.
3. Added option –zip-include which allows for additional files to be added to the zip file that contains the modules that implement the Python script. Thanks to Barray Warsaw for providing the initial patch.
4. Added support for handling read-only files properly. Thanks to Peter Grayson for pointing out the problem and providing a solution.
5. Added support for a frozen executable to be a symbolic link. Thanks to Robert Kiendl for providing the initial patch.
6. Enhanced the support for running a frozen executable that uses an existing Python installation to locate modules it requires. This is primarily of use for embedding Python where the interface is C but the ability to run from source is still desired.
7. Modified the documentation to indicate that building from source on Windows currently requires the mingw compiler (http://www.mingw.org).

8. Workaround the problem in Python 2.3 (fixed in Python 2.4) which causes a broken module to be left in sys.modules if an ImportError takes place during the execution of the code in that module. Thanks to Roger Binns for pointing this out.

Version 3.0 (September 2004)

1. Ensure that ldd is only run on extension modules.
2. Allow for using a compiler other than gcc for building the frozen base executables by setting the environment variable CC.
3. Ensure that the import lock is not held while executing the main script; otherwise, attempts to import a module within a thread will hang that thread as noted by Roger Binns.
4. Added support for replacing the paths in all frozen modules with something else (so that for example the path of the machine on which the freezing was done is not displayed in tracebacks)

Version 3.0 beta3 (September 2004)

1. Explicitly include the warnings module so that at runtime warnings are suppressed as when running Python normally.
2. Improve the extension loader so that an ImportError is raised when the dynamic module is not located; otherwise an error about missing attributes is raised instead.
3. Extension loaders are only created when copying dependencies since the normal module should be loadable in the situation where a Python installation is available.
4. Added support for Python 2.4.
5. Fixed the dependency checking for wxPython to be a little more intelligent.

Version 3.0 beta2 (July 2004)

1. Fix issues with locating the initscripts and bases relative to the directory in which the executable was started.
2. Added new base executable ConsoleKeepPath which is used when an existing Python installation is required (such as for FreezePython itself).
3. Forced the existence of a Python installation to be ignored when using the standard Console base executable.
4. Remove the existing file when copying dependent files; otherwise, an error is raised when attempting to overwrite read-only files.
5. Added option -O (or -OO) to FreezePython to set the optimization used when generating bytecode.

Version 3.0 beta1 (June 2004)

1. cx_Freeze now requires Python 2.3 or higher since it takes advantage of the ability of Python 2.3 and higher to import modules from zip files. This makes the freezing process considerably simpler and also allows for the execution of multiple frozen packages (such as found in COM servers or shared libraries) without requiring modification to the Python modules.
2. All external dependencies have been removed. cx_Freeze now only requires a standard Python distribution to do its work.

3. Added the ability to define the initialization scripts that cx_Freeze uses on startup of the frozen program. Previously, these scripts were written in C and could not easily be changed; now they are written in Python and can be found in the initscripts directory (and chosen with the new –init-script option to FreezePython).
4. The base executable ConsoleSetLibPath has been removed and replaced with the initscript ConsoleSetLibPath.
5. Removed base executables for Win32 services and Win32 COM servers. This functionality will be restored in the future but it is not currently in a state that is ready for release. If this functionality is required, please use py2exe or contact me for my work in progress.
6. The attribute sys.frozen is now set so that more recent pywin32 modules work as expected when frozen.
7. Added option –include-path to FreezePython to allow overriding of sys.path without modifying the environment variable PYTHONPATH.
8. Added option –target-dir/–install-dir to specify the directory in which the frozen executable and its dependencies will be placed.
9. Removed the option –shared-lib since it was used for building shared libraries and can be managed with the initscript SharedLib.py.
10. MakeFrozenBases.py now checks the platform specific include directory as requested by Michael Partridge.

Version 2.2 (August 2003)

1. Add option (–ext-list-file) to FreezePython to write the list of extensions copied to the installation directory to a file. This option is useful in cases where multiple builds are performed into the same installation directory.
2. Pass the arguments on the command line through to Win32 GUI applications. Thanks to Michael Porter for pointing this out.
3. Link directly against the python DLL when building the frozen bases on Windows, thus eliminating the need for building an import library.
4. Force sys.path to include the directory in which the script to be frozen is found.
5. Make sure that the installation directory exists before attempting to copy the target binary into it.
6. The Win32GUI base has been modified to display fatal errors in message boxes, rather than printing errors to stderr, since on Windows the standard file IO handles are all closed.

Version 2.1 (July 2003)

1. Remove dependency on Python 2.2. Thanks to Paul Moore for not only pointing it out but providing patches.
2. Set up the list of frozen modules in advance, rather than doing it after Python is initialized so that implicit imports done by Python can be satisfied. The bug in Python 2.3 that demonstrated this issue has been fixed in the first release candidate. Thanks to Thomas Heller for pointing out the obvious in this instance!
3. Added additional base executable (ConsoleSetLibPath) to support setting the LD_LIBRARY_PATH variable on Unix platforms and restarting the executable to put the new setting into effect. This is primarily of use in distributing wxPython applications on Unix where the shared library has an embedded RPATH value which can cause problems.
4. Small improvements of documentation based on feedback from several people.
5. Print information about the files written or copied during the freezing process.
6. Do not copy extensions when freezing if the path is being overridden since it is expected that a full Python installation is available to the target users of the frozen binary.

7. Provide meaningful error message when the wxPython library cannot be found during the freezing process.

Version 2.0

1. Added support for in process (DLL) COM servers using PythonCOM.
2. Ensured that the frozen flag is set prior to determining the full path for the program in order to avoid warnings about Python not being found on some platforms.
3. Added include file and resource file to the source tree to avoid the dependency on the Wine message compiler for Win32 builds.
4. Dropped the option –copy-extensions; this now happens automatically since the resulting binary is useless without them.
5. Added a sample for building a Win32 service.
6. Make use of improved modules from Python 2.3 (which function under 2.2)

Version 1.1

1. Fixed import error with C extensions in packages; thanks to Thomas Heller for pointing out the solution to this problem.
2. Added options to FreezePython to allow for the inclusion of modules which will not be found by the module finder (–include-modules) and the exclusion of modules which will be found by the module finder but should not be included (–exclude-modules).
3. Fixed typo in README.txt.

CHAPTER 6

Developing cx_Freeze

The source code can be found on Bitbucket.

Downloads and issue tracking are on Sourceforge.

Contents:

The project's code layout

- `cx_Freeze/` (Python files)
  - `freezer.py` - The core class for freezing code.
  - `finder.py` - Discovers what modules are required by the code
  - `hooks.py` - A collection of functions which are triggered automatically by `finder.py` when certain packages are included or not found.
  - `dist.py` - The classes and functions with which cx_Freeze *extends distutils*.
  - `windist.py` - Extends distutils to build Windows installer packages.
  - `main.py` - The code behind the *cxfreeze script*.
- `source/` (C files)
  - `util.c` - Compiled functions for cx_Freeze itself. Compiles to `cx_Freeze.util`. Most of the functions are used only on Windows.
  - `bases/` - The source of the base executables which are used to launch your Python applications. Different bases serve for different types of application on Windows (GUI, console application or service). The base executable calls the initscript, which in turn calls the user's code.
- `initscripts/` - Python scripts which set up the interpreter to run from frozen code, then load the code from the zip file and set it running.
- `samples/` - Examples of using cx_Freeze with a number of common modules.

- `doc/` - The Sphinx documentation.

CHAPTER 7

Licensing

- Copyright © 2007-2017, Anthony Tuininga.
- Copyright © 2001-2006, Computronix (Canada) Ltd., Edmonton, Alberta, Canada.
- All rights reserved.

NOTE: this license is derived from the Python Software Foundation License which can be found at http://www.python.org/psf/license

License for cx_Freeze 5.0.2

1. This LICENSE AGREEMENT is between the copyright holders and the Individual or Organization ("Licensee") accessing and otherwise using cx_Freeze software in source or binary form and its associated documentation.
2. Subject to the terms and conditions of this License Agreement, the copyright holders hereby grant Licensee a nonexclusive, royalty-free, world-wide license to reproduce, analyze, test, perform and/or display publicly, prepare derivative works, distribute, and otherwise use cx_Freeze alone or in any derivative version, provided, however, that this License Agreement and this notice of copyright are retained in cx_Freeze alone or in any derivative version prepared by Licensee.
3. In the event Licensee prepares a derivative work that is based on or incorporates cx_Freeze or any part thereof, and wants to make the derivative work available to others as provided herein, then Licensee hereby agrees to include in any such work a brief summary of the changes made to cx_Freeze.
4. The copyright holders are making cx_Freeze available to Licensee on an "AS IS" basis. THE COPYRIGHT HOLDERS MAKE NO REPRESENTATIONS OR WARRANTIES, EXPRESS OR IMPLIED. BY WAY OF EXAMPLE, BUT NOT LIMITATION, THE COPYRIGHT HOLDERS MAKE NO AND DISCLAIM ANY REPRESENTATION OR WARRANTY OF MERCHANTABILITY OR FITNESS FOR ANY PARTICULAR PURPOSE OR THAT THE USE OF CX_FREEZE WILL NOT INFRINGE ANY THIRD PARTY RIGHTS.
5. THE COPYRIGHT HOLDERS SHALL NOT BE LIABLE TO LICENSEE OR ANY OTHER USERS OF CX_FREEZE FOR ANY INCIDENTAL, SPECIAL, OR CONSEQUENTIAL DAMAGES OR LOSS AS A RESULT OF MODIFYING, DISTRIBUTING, OR OTHERWISE USING CX_FREEZE, OR ANY DERIVATIVE THEREOF, EVEN IF ADVISED OF THE POSSIBILITY THEREOF.

6. This License Agreement will automatically terminate upon a material breach of its terms and conditions.
7. Nothing in this License Agreement shall be deemed to create any relationship of agency, partnership, or joint venture between the copyright holders and Licensee. This License Agreement does not grant permission to use copyright holder's trademarks or trade name in a trademark sense to endorse or promote products or services of Licensee, or any third party.
8. By copying, installing or otherwise using cx_Freeze, Licensee agrees to be bound by the terms and conditions of this License Agreement.

Computronix® is a registered trademark of Computronix (Canada) Ltd.

CHAPTER 8

Indices and tables

- genindex
- modindex
- search

Index

Symbols

C