

Distributing Python Modules (Legacy version)

Authors:	Greg Ward, Anthony Baxter
Email:	distutils-sig@python.org

See also:

Distributing Python Modules

The up to date module distribution documentations

This document describes the Python Distribution Utilities (“Distutils”) from the module developer’s point of view, describing how to use the Distutils to make Python modules and extensions easily available to a wider audience with very little overhead for build/release/install mechanics.

Note: This guide only covers the basic tools for building and distributing extensions that are provided as part of this version of Python. Third party tools offer easier to use and more secure alternatives. Refer to the [quick recommendations section](#) in the Python Packaging User Guide for more information.

- [1. An Introduction to Distutils](#)
 - [1.1. Concepts & Terminology](#)
 - [1.2. A Simple Example](#)
 - [1.3. General Python terminology](#)
 - [1.4. Distutils-specific terminology](#)
- [2. Writing the Setup Script](#)
 - [2.1. Listing whole packages](#)
 - [2.2. Listing individual modules](#)
 - [2.3. Describing extension modules](#)
 - [2.4. Relationships between Distributions and Packages](#)
 - [2.5. Installing Scripts](#)
 - [2.6. Installing Package Data](#)
 - [2.7. Installing Additional Files](#)
 - [2.8. Additional meta-data](#)
 - [2.9. Debugging the setup script](#)
- [3. Writing the Setup Configuration File](#)
- [4. Creating a Source Distribution](#)
 - [4.1. Specifying the files to distribute](#)
 - [4.2. Manifest-related options](#)
- [5. Creating Built Distributions](#)
 - [5.1. Creating RPM packages](#)

- 5.2. Creating Windows Installers
 - 5.3. Cross-compiling on Windows
 - 5.4. Vista User Access Control (UAC)
- 6. The Python Package Index (PyPI)
 - 6.1. PyPI overview
 - 6.2. Distutils commands
 - 6.3. PyPI package display
- 7. Examples
 - 7.1. Pure Python distribution (by module)
 - 7.2. Pure Python distribution (by package)
 - 7.3. Single extension module
 - 7.4. Checking a package
 - 7.5. Reading the metadata
- 8. Extending Distutils
 - 8.1. Integrating new commands
 - 8.2. Adding new distribution types
- 9. Command Reference
 - 9.1. Installing modules: the **install** command family
 - 9.2. Creating a source distribution: the **sdist** command
- 10. API Reference
 - 10.1. `distutils.core` — Core Distutils functionality
 - 10.2. `distutils.ccompiler` — CCompiler base class
 - 10.3. `distutils.unixccompiler` — Unix C Compiler
 - 10.4. `distutils.msvccompiler` — Microsoft Compiler
 - 10.5. `distutils.bcppcompiler` — Borland Compiler
 - 10.6. `distutils.cygwincompiler` — Cygwin Compiler
 - 10.7. `distutils.archive_util` — Archiving utilities
 - 10.8. `distutils.dep_util` — Dependency checking
 - 10.9. `distutils.dir_util` — Directory tree operations
 - 10.10. `distutils.file_util` — Single file operations
 - 10.11. `distutils.util` — Miscellaneous other utility functions
 - 10.12. `distutils.dist` — The Distribution class
 - 10.13. `distutils.extension` — The Extension class
 - 10.14. `distutils.debug` — Distutils debug mode
 - 10.15. `distutils.errors` — Distutils exceptions
 - 10.16. `distutils.fancy_getopt` — Wrapper around the standard `getopt` module
 - 10.17. `distutils.filelist` — The FileList class
 - 10.18. `distutils.log` — Simple PEP 282-style logging
 - 10.19. `distutils.spawn` — Spawn a sub-process
 - 10.20. `distutils.sysconfig` — System configuration information
 - 10.21. `distutils.text_file` — The TextFile class
 - 10.22. `distutils.version` — Version number classes
 - 10.23. `distutils.cmd` — Abstract base class for Distutils commands
 - 10.24. Creating a new Distutils command

- 10.25. `distutils.command` — Individual Distutils commands
- 10.26. `distutils.command.bdist` — Build a binary installer
- 10.27. `distutils.command.bdist_packager` — Abstract base class for packagers
- 10.28. `distutils.command.bdist_dumb` — Build a “dumb” installer
- 10.29. `distutils.command.bdist_msi` — Build a Microsoft Installer binary package
- 10.30. `distutils.command.bdist_rpm` — Build a binary distribution as a Redhat RPM and SRPM
- 10.31. `distutils.command.bdist_wininst` — Build a Windows installer
- 10.32. `distutils.command.sdist` — Build a source distribution
- 10.33. `distutils.command.build` — Build all files of a package
- 10.34. `distutils.command.build_clib` — Build any C libraries in a package
- 10.35. `distutils.command.build_ext` — Build any extensions in a package
- 10.36. `distutils.command.build_py` — Build the `.py/.pyc` files of a package
- 10.37. `distutils.command.build_scripts` — Build the scripts of a package
- 10.38. `distutils.command.clean` — Clean a package build area
- 10.39. `distutils.command.config` — Perform package configuration
- 10.40. `distutils.command.install` — Install a package
- 10.41. `distutils.command.install_data` — Install data files from a package
- 10.42. `distutils.command.install_headers` — Install C/C++ header files from a package
- 10.43. `distutils.command.install_lib` — Install library files from a package
- 10.44. `distutils.command.install_scripts` — Install script files from a package
- 10.45. `distutils.command.register` — Register a module with the Python Package Index
- 10.46. `distutils.command.check` — Check the meta-data of a package