4. Creating a Source Distribution

As shown in section A Simple Example, you use the **sdist** command to create a source distribution. In the simplest case,

```
python setup.py sdist
```

(assuming you haven't specified any **sdist** options in the setup script or config file), **sdist** creates the archive of the default format for the current platform. The default format is a gzip'ed tar file (`.tar.gz`) on Unix, and ZIP file on Windows.

You can specify as many formats as you like using the `--formats` option, for example:

```
python setup.py sdist --formats=gztar,zip
```

to create a gzipped tarball and a zip file. The available formats are:

Format	Description	Notes
`zip`	zip file (`.zip`)	(1),(3)
`gztar`	gzip'ed tar file (`.tar.gz`)	(2)
`bztar`	bzip2'ed tar file (`.tar.bz2`)	
`xztar`	xz'ed tar file (`.tar.xz`)	
`ztar`	compressed tar file (`.tar.Z`)	(4)
`tar`	tar file (`.tar`)	

Changed in version 3.5: Added support for the `xztar` format.

Notes:

1. default on Windows
2. default on Unix
3. requires either external **zip** utility or `zipfile` module (part of the standard Python library since Python 1.6)
4. requires the **compress** program. Notice that this format is now pending for deprecation and will be removed in the future versions of Python.

When using any `tar` format (`gztar`, `bztar`, `xztar`, `ztar` or `tar`), under Unix you can specify the `owner` and `group` names that will be set for each member of the archive.

For example, if you want all files of the archive to be owned by root:

```
python setup.py sdist --owner=root --group=root
```

4.1. Specifying the files to distribute

If you don't supply an explicit list of files (or instructions on how to generate one), the **sdist** command puts a minimal default set into the source distribution:

- all Python source files implied by the `py_modules` and `packages` options
- all C source files mentioned in the `ext_modules` or `libraries` options
- scripts identified by the `scripts` option See Installing Scripts.
- anything that looks like a test script: `test/test*.py` (currently, the Distutils don't do anything with test scripts except include them in source distributions, but in the future there will be a standard for testing Python module distributions)
- `README.txt` (or `README`), `setup.py` (or whatever you called your setup script), and `setup.cfg`
- all files that matches the `package_data` metadata. See Installing Package Data.
- all files that matches the `data_files` metadata. See Installing Additional Files.

Sometimes this is enough, but usually you will want to specify additional files to distribute. The typical way to do this is to write a *manifest template*, called `MANIFEST.in` by default. The manifest template is just a list of instructions for how to generate your manifest file, `MANIFEST`, which is the exact list of files to include in your source distribution. The **sdist** command processes this template and generates a manifest based on its instructions and what it finds in the filesystem.

If you prefer to roll your own manifest file, the format is simple: one filename per line, regular files (or symlinks to them) only. If you do supply your own `MANIFEST`, you must specify everything: the default set of files described above does not apply in this case.

Changed in version 3.1: An existing generated `MANIFEST` will be regenerated without **sdist** comparing its modification time to the one of `MANIFEST.in` or `setup.py`.

Changed in version 3.1.3: `MANIFEST` files start with a comment indicating they are generated. Files without this comment are not overwritten or removed.

Changed in version 3.2.2: **sdist** will read a `MANIFEST` file if no `MANIFEST.in` exists, like it used to do.

The manifest template has one command per line, where each command specifies a set of files to include or exclude from the source distribution. For an example, again we turn to the Distutils' own manifest template:

```
include *.txt
recursive-include examples *.txt *.py
prune examples/sample?/build
```

The meanings should be fairly clear: include all files in the distribution root matching `*.txt`, all files anywhere under the `examples` directory matching `*.txt` or `*.py`, and exclude all directories matching `examples/sample?/build`. All of this is done *after* the standard include set, so you can exclude files from the standard set with explicit instructions in the manifest template. (Or, you can use the `--no-defaults` option to disable the standard set entirely.) There are several other commands available in the manifest template mini-language; see section Creating a source distribution: the sdist command.

The order of commands in the manifest template matters: initially, we have the list of default files as described above, and each command in the template adds to or removes from that list of files. Once we have fully processed the manifest template, we remove files that should not be included in the source distribution:

- all files in the Distutils "build" tree (default `build/`)
- all files in directories named `RCS`, `CVS`, `.svn`, `.hg`, `.git`, `.bzr` or `_darcs`

Now we have our complete list of files, which is written to the manifest for future reference, and then used to build the source distribution archive(s).

You can disable the default set of included files with the `--no-defaults` option, and you can disable the standard exclude set with `--no-prune`.

Following the Distutils' own manifest template, let's trace how the **sdist** command builds the list of files to include in the Distutils source distribution:

1. include all Python source files in the `distutils` and `distutils/command` subdirectories (because packages corresponding to those two directories were mentioned in the `packages` option in the setup script—see section Writing the Setup Script)
2. include `README.txt`, `setup.py`, and `setup.cfg` (standard files)
3. include `test/test*.py` (standard files)
4. include `*.txt` in the distribution root (this will find `README.txt` a second time, but such redundancies are weeded out later)
5. include anything matching `*.txt` or `*.py` in the sub-tree under `examples`,
6. exclude all files in the sub-trees starting at directories matching `examples/sample?/build`—this may exclude files included by the previous two steps, so it's important that the `prune` command in the manifest template comes after the `recursive-include` command
7. exclude the entire `build` tree, and any `RCS`, `CVS`, `.svn`, `.hg`, `.git`, `.bzr` and `_darcs` directories

Just like in the setup script, file and directory names in the manifest template should always be slash-separated; the Distutils will take care of converting them to the standard representation on your platform. That way, the manifest template is portable across operating systems.

4.2. Manifest-related options

The normal course of operations for the **sdist** command is as follows:

- if the manifest file (`MANIFEST` by default) exists and the first line does not have a comment indicating it is generated from `MANIFEST.in`, then it is used as is, unaltered
- if the manifest file doesn't exist or has been previously automatically generated, read `MANIFEST.in` and create the manifest
- if neither `MANIFEST` nor `MANIFEST.in` exist, create a manifest with just the default file set
- use the list of files now in `MANIFEST` (either just generated or read in) to create the source distribution archive(s)

There are a couple of options that modify this behaviour. First, use the `--no-defaults` and `--no-prune` to disable the standard "include" and "exclude" sets.

Second, you might just want to (re)generate the manifest, but not create a source distribution:

```
python setup.py sdist --manifest-only
```

`-o` is a shortcut for `--manifest-only`.