6. The Python Package Index (PyPI)

The Python Package Index (PyPI) stores meta-data describing distributions packaged with distutils, as well as package data like distribution files if a package author wishes.

Distutils provides the **register** and **upload** commands for pushing meta-data and distribution files to PyPI, respectively. See Distutils commands for information on these commands.

6.1. PyPI overview

PyPI lets you submit any number of versions of your distribution to the index. If you alter the meta-data for a particular version, you can submit it again and the index will be updated.

PyPI holds a record for each (name, version) combination submitted. The first user to submit information for a given name is designated the Owner of that name. Changes can be submitted through the **register** command or through the web interface. Owners can designate other users as Owners or Maintainers. Maintainers can edit the package information, but not designate new Owners or Maintainers.

By default PyPI displays only the newest version of a given package. The web interface lets one change this default behavior and manually select which versions to display and hide.

For each version, PyPI displays a home page. The home page is created from the `long_description` which can be submitted via the **register** command. See PyPI package display for more information.

6.2. Distutils commands

Distutils exposes two commands for submitting package data to PyPI: the register command for submitting meta-data to PyPI and the upload command for submitting distribution files. Both commands read configuration data from a special file called a .pypirc file.

6.2.1. The `register` command

The distutils command **register** is used to submit your distribution's meta-data to an index server. It is invoked as follows:

```
python setup.py register
```

Distutils will respond with the following prompt:

```
running register
We need to know who you are, so please choose either:
    1. use your existing login,
    2. register as a new user,
    3. have the server generate a new password for you (and email it t
    4. quit
Your selection [default 1]:
```

Note: if your username and password are saved locally, you will not see this menu. Also, refer to The .pypirc file for how to store your credentials in a `.pypirc` file.

If you have not registered with PyPI, then you will need to do so now. You should choose option 2, and enter your details as required. Soon after submitting your details, you will receive an email which will be used to confirm your registration.

Once you are registered, you may choose option 1 from the menu. You will be prompted for your PyPI username and password, and **register** will then submit your meta-data to the index.

See Additional command options for options to the **register** command.

6.2.2. The `upload` command

The distutils command **upload** pushes the distribution files to PyPI.

The command is invoked immediately after building one or more distribution files. For example, the command

```
python setup.py sdist bdist_wininst upload
```

will cause the source distribution and the Windows installer to be uploaded to PyPI. Note that these will be uploaded even if they are built using an earlier invocation of `setup.py`, but that only distributions named on the command line for the invocation including the **upload** command are uploaded.

If a **register** command was previously called in the same command, and if the password was entered in the prompt, **upload** will reuse the entered password. This is useful if you do not want to store a password in clear text in a `.pypirc` file.

You can use the `--sign` option to tell **upload** to sign each uploaded file using GPG (GNU Privacy Guard). The **gpg** program must be available for execution on the system `PATH`. You can also specify which key to use for signing using the `--identity=name` option.

See Additional command options for additional options to the **upload** command.

6.2.3. Additional command options

This section describes options common to both the **register** and **upload** commands.

The `--repository` or `-r` option lets you specify a PyPI server different from the default. For example:

```
python setup.py sdist bdist_wininst upload -r https://example.com/pypi
```

For convenience, a name can be used in place of the URL when the `.pypirc` file is configured to do so. For example:

```
python setup.py register -r other
```

See The .pypirc file for more information on defining alternate servers.

The `--show-response` option displays the full response text from the PyPI server, which is useful when debugging problems with registering and uploading.

6.2.4. The `.pypirc` file

The **register** and **upload** commands both check for the existence of a `.pypirc` file at the location `$HOME/.pypirc`. If this file exists, the command uses the username, password, and repository URL configured in the file. The format of a `.pypirc` file is as follows:

```
[distutils]
index-servers =
    pypi

[pypi]
repository: <repository-url>
username: <username>
password: <password>
```

The *distutils* section defines an *index-servers* variable that lists the name of all sections describing a repository.

Each section describing a repository defines three variables:

- *repository*, that defines the url of the PyPI server. Defaults to `https://upload.pypi.org/legacy/`.
- *username*, which is the registered username on the PyPI server.
- *password*, that will be used to authenticate. If omitted the user will be prompt to type it when needed.

If you want to define another server a new section can be created and listed in the *index-servers* variable:

```
[distutils]
index-servers =
    pypi
    other

[pypi]
repository: <repository-url>
username: <username>
password: <password>

[other]
repository: https://example.com/pypi
username: <username>
password: <password>
```

This allows the **register** and **upload** commands to be called with the `--repository` option as described in Additional command options.

Specifically, you might want to add the PyPI Test Repository to your `.pypirc` to facilitate testing before doing your first upload to `PyPI` itself.

6.3. PyPI package display

The `long_description` field plays a special role at PyPI. It is used by the server to display a home page for the registered package.

If you use the reStructuredText syntax for this field, PyPI will parse it and display an HTML output for the package home page.

The `long_description` field can be attached to a text file located in the package:

```python
from distutils.core import setup

with open('README.txt') as file:
    long_description = file.read()

setup(name='Distutils',
      long_description=long_description)
```

In that case, `README.txt` is a regular reStructuredText text file located in the root of the package besides `setup.py`.

To prevent registering broken reStructuredText content, you can use the **rst2html** program that is provided by the `docutils` package and check the `long_description` from the command line:

```
$ python setup.py --long-description | rst2html.py > output.html
```

`docutils` will display a warning if there's something wrong with your syntax. Because PyPI applies additional checks (e.g. by passing `--no-raw` to `rst2html.py` in the command above), being able to run the command above without warnings does not guarantee that PyPI will convert the content successfully.