# How to Set Up & Publish a Package Using UV on PyPI

**The Working Way - Video: [Publish Your Own Python Package in Less Than 10 Minutes](https://www.youtube.com/watch?v=rM1SqONoPXs)**

Summary:

Let UV handle the package structure for you. For example, me & many developers didn’t initially know that a src folder is required. This folder will contain your package files.

## **1.** Initialize the Package

Command: uv init --package <package-name>

This creates a folder with the package name. Inside, you'll find:

- pyproject.toml file

- README.md

- .python-version

- src/ folder (place your original package/module files here)

Delete the content inside \_\_init\_\_.py and replace it with the exposure of your modules/functions using the \_\_all\_\_ list.

\*\**Important Note:*\*\*: Delete all previously created \_\_pycache\_\_ files in your original package before moving it into src/, as they might cause issues.

\*\**Important Note*:\*\* Move inside the package's directory. Do not work from the root if you are about to build and publish.

## **2.** Edit the pyproject.toml File

- Add dependencies (if you did not use UV instead of pip from the start).

- Add your documentation and repository links under [project.links].

- Remove the main script from the pyproject.toml file if not needed.

- Generate the README for the PyPI page.

The README should contain:

- Description: At the very top, with a preview.

- Detailed package information for the PyPI page.

- Properly listed dependencies to allow for correct downloads.

- [project.urls]: Links to Home, Repo, and Documentation (later).

\*\**Extra Note*:\*\* To generate an old-style requirements.txt file, run:

uv pip install -r requirements.txt

## **3.** Build the Package Locally (within the package’s directory)

Command: uv build --no-sources

- Creates the dist folder with .whl and .tar files.

- Sometimes, you may find the dist folder on the desktop. If so, move it into the root folder inside the package manually.

The --no-sources flag tells UV to build only binary distributions (like wheels) and exclude source distributions (like .tar.gz). This is common for simple pure-Python packages or when you want to avoid distributing source code.

## **4.** Publish on TestPyPI

Command:

uv publish --token pypi-<your-api-key> --publish-url https://test.pypi.org/legacy/

\*\*Extra:\*\* Summary of Required Steps to Publish a New Version:

- Bump the version in pyproject.toml.

- Build with uv build --no-sources.

- Publish with uv publish --token ... --publish-url https://test.pypi.org/legacy/.

Note:

After updating your package version in pyproject.toml, always delete old build files from the dist/ directory before rebuilding and publishing. This ensures you upload the correct, latest version to PyPI or TestPyPI.

Note - Steps:

1. Delete all files in dist/ (e.g., attendance\_tooltt-0.1.0\*).

2. Run uv build --no-sources to create new build files.

3. Publish the new version with uv publish --token <your-api-key> --publish-url https://test.pypi.org/legacy/.

This prevents version conflicts and upload errors!

Additional Information:

- The src folder is a common convention to separate source code from other files.

- Inside src, your actual package is attendance\_tool\_msp, which contains your Python modules and (optionally) assets for GUI.

- Your build and configuration files are in attendance\_tool\_msp (used for the package name, installable through pip, dependencies, etc.).

- Your actual Python package is in src/attendance\_tool\_msp, and this will be the part downloaded by the user.

## **5. Publish On PyPi**

Command: uv publish --token pypi-<your-api-key>

Only Publish On PyPi when you’re 100% satisfied through testing on TestPyPi.