

x_make_pypi_x “ Release Foundry Manual

This foundry crafts the wheels and sdists we ship to PyPI. It mirrors the target module into a sterile workspace, injects typing artefacts, builds with `python -m build`, and sends the payload through `twine` with credential checks and audit logging. When the orchestrator orders a release, this module produces the evidence.

Mission Log

- Clone the package into an isolated build directory to shield source trees.
- Emit `py.typed`, stub files, and `MANIFEST.in` entries so consumers receive full typing signals.
- Build wheels and sdists, probe PyPI metadata to avoid duplicate releases, and upload via `twine` with `--skip-existing` and strict credential handling.
- Register artefact manifests and checksums in `make_all_summary.json` so the Release Assembly column shows the exact files that left the lab.

Instrumentation

- Python 3.11 or newer.
- `build` and `twine` installed in the active environment.
- Optional QA: Ruff, Black, MyPy, Pyright, pytest for local verification.
- Credentials via `.pyprc` or `TWINE_*` environment variables (`TWINE_API_TOKEN`, `TWINE_USERNAME`, `TWINE_PASSWORD`).

Operating Procedure

1. `python -m venv .venv`
2. `\\.venv\\Scripts\\Activate.ps1`
3. `python -m pip install --upgrade pip`
4. `pip install build twine`
5. Export credentials if `.pyprc` is not configured: `powershell $env:TWINE_API_TOKEN = "pypi-your-token"`
6. Drive publishing from the orchestrator or a focused script. Example driver: `python from x_cls_make_pypi_x import XClsMakePypiX`

```
publisher = XClsMakePypiX( name="your_package", version="0.1.0", author="Control Room Ops", email="lab.ops@example.com",
description="Blueprint for a PyPI release", license_text="MIT", dependencies=["requests"], ctx=None, ) publisher.prepare_and_publish(
main_file="src/your_package/init.py", ancillary_files=["README.md", "pyproject.toml"], ) ``
```

Evidence Checks

Check	Command	--- ---	Formatting sweep	Lint interrogation	Type audit
	<code>python -m mypy x_cls_make_pypi_x.py</code>		<code>python -m black .</code>	<code>python -m ruff check .</code>	<code>python -m ruff check .</code>
			<code>python -m pyright</code>	<code>python -m ruff check .</code>	<code>python -m ruff check .</code>
				<code>python -m ruff check .</code>	<code>python -m ruff check .</code>

Reconstitution Drill

During the monthly rebuild I reinstall `build` and `twine`, execute a dry-run upload to TestPyPI, and confirm the artefact manifest still wires into the orchestrator summary. Tool versions, runtimes, and credential checks get logged; any deviation triggers documentation updates and Change Control entries.

Conduct Code

Publish from a clean workspace, document the package intent and hashes in Change Control, and never retry a failed upload until the evidence trail is corrected. PyPI rejections are signals, not suggestions.

Sole Architect's Note

I designed and maintain this release foundry end to end “workspace isolation, stub emitters, credential guards, logging, orchestrator integration. Years of release engineering let me run the entire pipeline without external approvals.

Legacy Staffing Estimate

- Conventional replication would require: 1 release engineer, 1 packaging specialist, 1 security analyst, 1 technical writer.
- Timeline: 13 “15 engineer-weeks for parity.
- Budget: USD 110 “140k plus ongoing compliance maintenance and credential rotation.

Technical Footprint

- Language & Tooling: Python 3.11+, build, twine, importlib.metadata, pathlib workspace management.
- Security Surface: .pypirc integration, TWINE_* environment controls, JSON manifests for Change Control.
- Quality Net: Ruff, Black, MyPy, Pyright, optional pytest harnesses.
- Integrations: Orchestrator release stage, dependency alignment via x_make_pip_updates_x, logging from x_make_common_x.