Package: pyro (via r-universe)

July 14, 2026

Title Manage Python Environments in R with uv

Version 0.1.1

Description Provides uv-backed Python virtual-environment management for the fyr ecosystem (reportifyr, presentifyr R packages).

License GPL (>= 3)

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Depends R (>= 4.1.0)

Imports here, log4r, processx, rlang

Suggests knitr, mockery, rmarkdown, testthat (>= 3.0.0), withr

VignetteBuilder knitr

Config/testthat/edition 3

Repository https://a2-ai.r-universe.dev

Date/Publication 2026-06-22 14:43:34 UTC

RemoteUrl https://github.com/a2-ai/pyro

RemoteRef HEAD

RemoteSha 66065c19a9aad1cf2cc4718aee9da013e6a3209f

Contents

get_proj_dir *Resolve the project root directory*

Description

Single canonical resolver for "where does this project live?". Used as the default for `pyproject_dir` and `venv_dir` in `initialize_python()`, and as a base for sibling-package wrappers that need to locate the project's `pyproject.toml`.

Usage

```
get_proj_dir()
```

Details

Resolution order:

1. `getOption("venv_dir")` if set (cached project root, written by prior calls to `initialize_python()`).
2. `here::here()` otherwise.

Value

Absolute path to the project root.

Examples

```
## Not run:
get_proj_dir()

## End(Not run)
```

get_uv_path *Locate the uv binary*

Description

Searches `PATH` first, then known install locations (`~/.local/bin/uv`, `~/.cargo/bin/uv`) cross-platform. Respects the `HOME` environment variable for test isolation.

Usage

```
get_uv_path(quiet = FALSE)
```

Arguments

`quiet`	If `TRUE`, suppress the log message when uv is not found.

Details

Use this to detect whether uv is installed without triggering an install. `initialize_python()` performs the install; this helper is a pure lookup.

Value

Path to the uv executable, or `NULL` if none is found.

Examples

```
## Not run:
uv_path <- get_uv_path()
if (is.null(uv_path)) message("uv not installed yet")

## End(Not run)
```

`get_uv_version` *Get the installed version of uv*

Description

Shells out to uv `--version` and parses the result. Useful for diagnostics and for callers that need to gate behavior on the uv version (e.g. only set a flag introduced in a specific release).

Usage

```
get_uv_version(uv_path)
```

Arguments

`uv_path`	Path to the uv executable. Typically obtained from `get_uv_path()`.

Value

Character scalar of the uv version (e.g., `"0.7.8"`).

Examples

```
## Not run:
uv_path <- get_uv_path()
if (!is.null(uv_path)) get_uv_version(uv_path)

## End(Not run)
```

get_venv_uv_paths *Resolve paths to the uv binary and the project's* .venv *directory*

Description

Looks up the project root via `get_proj_dir()` (which reads `getOption("venv_dir")`, falling back to `here::here()`) and caches the resolved root in `options("venv_dir")` so subsequent calls are cheap.

Usage

```
get_venv_uv_paths()
```

Value

Named list with `uv` (path to uv) and `venv` (path to the `.venv/` directory).

Examples

```
## Not run:
get_venv_uv_paths()

## End(Not run)
```

initialize_python *Initialize the uv-managed Python virtual environment*

Description

Ensures uv is installed at `uv_version`, then locks and syncs the project's `.venv/` against `<pyproject_dir>/pyproject.toml`. If no `pyproject.toml` exists at `pyproject_dir`, one is seeded from pyro's bundled spec (templated with the project's directory name) before lock and sync.

Usage

```
initialize_python(
  continue = NULL,
  venv_dir = get_proj_dir(),
  uv_version = getOption("uv.version"),
  groups = NULL,
  pyproject_dir = NULL
)
```

Arguments

continue	Optional. "Y" / "n" or TRUE / FALSE to bypass the interactive prompt. The same prompt covers both the uv install and the seed step (when the project has no pyproject.toml yet).
venv_dir	Parent directory of .venv/. Defaults to get_proj_dir().
uv_version	Version of uv to install. Defaults to getOption("uv.version"). When NULL (option unset), behavior is: if uv is already installed, its existing version is used (no reinstall); otherwise "0.7.8" is installed as a conservative fallback.
groups	Character vector of dependency-group names from the project's pyproject.toml (e.g. "reportifyr", "presentifyr"). When NULL (default), every declared group is installed via uv sync --frozen --all-groups. When supplied, uv runs in additive mode (uv sync --frozen --inexact --group <g> ...) so sibling fyr-package calls coexist without removing each other's deps. Group names are validated by uv; an unknown group surfaces uv's error.
pyproject_dir	Directory containing the project's pyproject.toml. Defaults to venv_dir. Override for non-default project layouts (LLM agents, monorepos, closed-source projects with secret toml elsewhere). The directory must already exist; pyro will not create it.

Details

The project's pyproject.toml is the source of truth. pyro seeds it on first call but never modifies an existing toml, group upserts are the responsibility of sibling-package wrappers via write_group_to_pyproject().

Value

Invisibly NULL.

Examples

```
## Not run:
initialize_python()                              # all groups
initialize_python(groups = "reportifyr")        # only reportifyr's group
initialize_python(groups = "presentifyr")       # only presentifyr's group
initialize_python(pyproject_dir = "~/proj")     # custom project root

## End(Not run)
```

---

pyro *pyro: uv-backed Python Environment Helpers for the fyr Ecosystem*

Description

Provides uv-backed Python virtual-environment management used across the fyr ecosystem (reportifyr, presentifyr). Wraps the uv CLI via processx for locating the uv binary, bootstrapping a .venv/, installing pinned Python dependencies, and running Python scripts.

Public API

All exported functions are part of the public API and may be called directly by sibling fyr packages and third-party projects.

- initialize_python: Bootstraps uv (if missing), creates a .venv/, and installs the requested Python packages.
- write_group_to_pyproject: Idempotent merge helper for sibling and third-party packages to wire their Python dependency group into the project's pyproject.toml.
- get_proj_dir: Project-root resolver (reads getOption("venv_dir") then falls back to here::here()).
- get_venv_uv_paths: Resolves the paths to the uv binary and the project's .venv/ directory. Used by host packages before invoking Python scripts.
- run_python_script: Invokes a Python script inside the uv-managed venv via processx.
- get_uv_path, get_uv_version: uv binary resolution and version detection.

Logging

pyro writes log lines through an internal log4r logger whose threshold is read from getOption("pyro.verbose", "WARN") on each emission. To change verbosity, set the option options(pyro.verbose = "DEBUG").
No reload or toggle call is required; the next log call reflects the new level. Use withr::with_options(list(pyro.verbose = "DEBUG"), ...) to scope a verbose level to a single block.

Author(s)

Maintainer: Jacob Dumbleton <jacob@a2-ai.com>

Authors:

- Matthew Smith <matthews@a2-ai.com>

---

run_python_script *Run a Python script inside the uv-managed venv*

Description

Shells out to uv via processx::run() with the venv's VIRTUAL_ENV and (optionally) a caller-supplied PYTHONPATH. pyro does not persist subprocess output to any file — routing of the Python subprocess' stderr is entirely the caller's concern. Supply stderr_callback to capture, reformat, or filter stderr lines; wrap this call in tryCatch() if you want to act on a subprocess crash.

Usage

```
run_python_script(
  uv_path,
  args,
  venv_path,
  script_name,
  pythonpath = NULL,
```

```
  stderr_callback = NULL,
  verbose_env = NULL
)
```

Arguments

uv_path	Path to the uv executable.
args	Arguments to pass to uv (e.g., c("run", "-m", "my_cli")).
venv_path	Path to the .venv/ directory (sets VIRTUAL_ENV).
script_name	Human-readable label used in the error message.
pythonpath	Optional directory to expose as PYTHONPATH (for host packages whose Python modules live under their inst/python/).
stderr_callback	Optional function(chunk, proc) callback passed to processx::run() as stderr_callback. If NULL, stderr is streamed to the console unchanged via cat().
verbose_env	Optional name of an environment variable to surface to the subprocess (for example, the caller's verbosity control).

Value

The result from processx::run().

Examples

```
## Not run:
paths <- pyro::get_venv_uv_paths()
pyro::run_python_script(
  uv_path     = paths$uv,
  venv_path   = paths$venv,
  args        = c("run", "-m", "my_module"),
  script_name = "my_module"
)

## End(Not run)
```

---

write_group_to_pyproject

Ensure a dependency group is present in the project's pyproject.toml

---

Description

Idempotent merge helper called by sibling-package wrappers (reportifyr, presentifyr) and third-party apps to wire their Python dependency group into the project's pyproject.toml.

Usage

```
write_group_to_pyproject(name, deps = NULL, pyproject_dir = get_proj_dir())
```

Arguments

name	Character group name (e.g. "reportifyr").
deps	Optional character vector of dependency strings (e.g. c("pillow==11.1.0", "requests==2.31.0")). When NULL (default), the bundled pyproject.toml is consulted for name; if name is not bundled, an error is raised.
pyproject_dir	Directory containing the project's pyproject.toml. Defaults to get_proj_dir().

Details

Merge semantics by case:

No pyproject.toml No-op, returns FALSE. Seeding is initialize_python()'s job; this helper composes with it but does not duplicate it.

No [dependency-groups] **table** Append the table containing name. Returns TRUE.

Table exists, no name **subgroup** Append the subgroup with deps. Returns TRUE.

Subgroup exists, dep names match Per-dep version comparison. If versions match: no-op, returns FALSE. If versions differ: keep the user's pin (do not overwrite), emit a one-line message per diff. Returns FALSE.

Subgroup exists, dep names differ Add any deps missing from the project's group with the spec's pins. Never remove deps the user added. Returns TRUE.

The "spec" is pyro's bundled pyproject.toml for fyr-blessed groups (when deps = NULL), or the caller-supplied deps for third-party apps.

Value

Invisibly, TRUE if the toml was mutated, FALSE otherwise.

Examples

```
## Not run:
# fyr-blessed group: pins read from bundled spec
write_group_to_pyproject("reportifyr")

# third-party app: pins supplied explicitly
write_group_to_pyproject("myapp", deps = c("requests==2.31.0"))

## End(Not run)
```

Index