Package 'tarpolyglot'

September 3, 2026

Type Package

Title Run Python, Julia, and Rust Inside 'targets' Pipeline Steps

Version 0.2.1

Description Adds target constructors that make it easy to use Python, Julia, and Rust inside a 'targets' pipeline using 'reticulate', 'JuliaCall', and 'rextendr'. Provides tar_target_py(), tar_target_jl(), and tar_target_rs() (with matching _raw() variants), each mirroring 'targets::tar_target()' and 'targets::tar_target_raw()'. Python and Julia steps run a script via a live interpreter with optional R pre- and post-scripts; Rust steps compile '#[extendr]' functions and call them from an R post-script. Results are returned either as converted R objects or as files written to disk (format = ``file''). Dynamic branching, environment/version selection, a 'crew' controller for isolation, and the full set of tar_target_raw() arguments are supported.

License MIT + file LICENSE

URL https://github.com/Pierre9344/tarpolyglot, https://pierre9344.github.io/tarpolyglot/

BugReports https://github.com/Pierre9344/tarpolyglot/issues

Encoding UTF-8

Imports targets, reticulate, JuliaCall, crew, rextendr

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

Config/testthat/edition 3

VignetteBuilder knitr

SystemRequirements The external language toolchains are optional and each is only required by its matching targets constructor: a Python installation for tar_target_py(), a Julia installation for tar_target_jl(), and a Rust toolchain with Cargo for tar_target_rs() (on Windows, the GNU toolchain, to match R's ABI). A pipeline that uses a single language needs only that language's toolchain, and a pipeline that uses none of these constructors needs no external toolchain at all.

Config/roxygen2/version 8.1.0

NeedsCompilation no

Author Pierre Solomon [aut, cre] (ORCID:
<https://orcid.org/0000-0001-7187-2664>)

Maintainer Pierre Solomon <pierre.solomon@laposte.net>

Repository CRAN

Date/Publication 2026-09-03 09:50:03 UTC

Contents

`polyglot_controller` *A crew controller preconfigured for polyglot isolation*

Description

Returns a `crew::crew_controller_local()` tuned for pipelines that run Python or Julia steps. Because reticulate and JuliaCall embed a **single interpreter per process** (see `vignette("get_started", package = "tarpolyglot")`), the only way to get a fresh interpreter per target, and to use different environments/versions across targets, is process-level isolation. Passing this controller to `targets::tar_option_set()` sends targets to separate worker processes; with `tasks_max = 1` each worker is retired after one task, so every foreign step gets a clean interpreter that is torn down when the task finishes.

Usage

```
polyglot_controller(workers = 2L, tasks_max = 1L, seconds_idle = 30, ...)
```

Arguments

workers	Integer number of parallel worker processes. Default 2.
tasks_max	Integer max tasks a worker runs before it is retired (and its embedded interpreter torn down). Default 1 for maximum isolation: a fresh Python/Julia per target. Raise it (or set Inf) to reuse interpreters for throughput, accepting that same-language steps on a worker then share the foreign global namespace and one environment.
seconds_idle	Numeric seconds an idle worker waits before shutting down. Default 30.
...	Further arguments passed to crew::crew_controller_local(), e.g. seconds_timeout, garbage_collection, or reset_globals.

Details

This is the recommended default for polyglot pipelines. Put it near the top of _targets.R:

```
targets::tar_option_set(controller = tarpolyglot::polyglot_controller())
```

On a system where crew cannot build a controller at all, this function fails with an explanation instead of passing the underlying error through. The case seen in practice is a nanonext installed without TLS support: crew self-tests nanonext's TLS layer whenever a controller is constructed, even though this controller requests no TLS, and that self-test reports "Not supported". The remedy is to reinstall nanonext on a system where TLS support is available, or to run the pipeline with no controller at all, which costs only the per-step interpreter isolation described above.

Value

A crew controller object, ready for targets::tar_option_set(controller = ...).

See Also

tar_target_py(), tar_target_jl()

Examples

```
## Not run:
# _targets.R
library(targets)
library(tarpolyglot)
tar_option_set(controller = polyglot_controller(workers = 4))
list(
  tar_target_py(x, "scripts/step.py", retrieve = "result")
)

## End(Not run)
```

`run_jl_step` *Execute a Julia step (worker behind tar_target_jl)*

Description

Julia counterpart of `run_py_step()`. Runs, inside a fresh R environment, an optional R pre-script, a Python script (via **JuliaCall**). This is the function the target built by `tar_target_jl()` calls; it is exported so that call resolves at pipeline run time but this function is not destined to be called directly by the package users.

Usage

```
run_jl_step(
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = list(),
  output = "object",
  retrieve = NULL,
  files = NULL,
  julia_version = NULL,
  julia_home = getOption("tarpolyglot.julia_home"),
  julia_project = NULL,
  julia_packages = NULL
)
```

Arguments

`script`	Path to the Julia script to run (required).
`pre_script`	Optional path to an R script run before the Julia script. It is evaluated in the step environment, which already holds the named `inputs`. To hand objects to Julia, assign a named list `to_jl` in this script; each element is `julia_assign()`ed as a variable in Julia's `Main` module.
`post_script`	Optional path to an R script run after the Julia script. It is evaluated in the same environment, which now also holds `jl_get(name)` and `jl_call(fn, ...)`. In `output = "object"` mode the value of its last expression becomes the target value; in `output = "file"` mode it must return a character vector of file paths.
`inputs`	Named list of upstream target values, bound by name into the step environment. Supplied automatically by the constructor.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`retrieve`	Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.
`files`	Optional character vector of file paths to return when no `post_script` is supplied in file mode.

julia_version — Optional Julia version to select (e.g. "1.11"), used when julia_home is not given. Resolved to a juliaup-managed install. Default NULL uses the computer/global default Julia.

julia_home, julia_project, julia_packages — Julia environment selection. julia_home is the directory containing the julia executable (defaults to getOption("tarpolyglot.julia_home"); when unset and no julia_version, JuliaCall discovers Julia on PATH). julia_project is a Julia project environment (folder with Project.toml / Manifest.toml) to Pkg.activate(); when NULL, Julia's default global environment (@v#.#) is used. julia_packages is a character vector of packages to using before the script. The requested environment takes priority over an ambient JULIA_PROJECT environment variable (e.g. one set by an RStudio project config and inherited by crew workers): it is cleared for the duration of the Julia binding, so an explicit julia_project (or the global environment you get when none is given) wins over it, rather than JULIA_PROJECT silently selecting a different project. (JULIA_HOME is not cleared: it is a supported way to point at the default Julia.)

Value

The converted R object (object mode) or a character vector of normalised file paths (file mode).

See Also

run_py_step(), tar_target_jl()

Examples

```
## Not run:
# Normally invoked by tar_target_jl(); shown here as a direct call.
# scripts/sum.jl assigns `result`; pre.R builds `to_jl <- list(x = x)`.
run_jl_step(
  script = "scripts/sum.jl",
  pre_script = "scripts/pre.R",
  inputs = list(x = c(1, 2, 3)),
  retrieve = "result"
)

## End(Not run)
```

---

run_py_step *Execute a Python step (worker behind tar_target_py)*

Description

Runs, inside a fresh R environment, an optional R pre-script, a Python script (via **reticulate**), and an optional R post-script, then returns either a converted R object or a character vector of files. This is the function the target built by tar_target_py() calls; it is exported so that call resolves at pipeline run time but this function is not destined to be called directly by the package users.

Usage

```
run_py_step(
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = list(),
  output = "object",
  retrieve = NULL,
  files = NULL,
  python_version = NULL,
  env = NULL,
  env_manager = "system",
  python = NULL
)
```

Arguments

`script`	Path to the Python script to run (required).
`pre_script`	Optional path to an R script run before the Python script. It is evaluated in the step environment, which already holds the named `inputs`. To hand objects to Python, assign a named list `to_py` in this script; each element is pushed as a top-level variable in the Python `__main__` module.
`post_script`	Optional path to an R script run after the Python script. It is evaluated in the same environment, which now also holds `py` (the reticulate `__main__` module proxy) and `py_get(name)`. In `output = "object"` mode the value of its last expression becomes the target value; in `output = "file"` mode it must return a character vector of file paths.
`inputs`	Named list of upstream target values, bound by name into the step environment. Supplied automatically by the constructor.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`retrieve`	Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.
`files`	Optional character vector of file paths to return when no `post_script` is supplied in file mode.
`python_version`	Optional Python version to select (e.g. `"3.12"` or `">=3.11"`), used only when no environment and no explicit `python` path are given. reticulate fetches/selects it (via its uv-backed ephemeral environment) with `reticulate::py_require()`. Default `NULL` uses the computer/global default Python. Use this when you only care about the interpreter *version* and are happy for reticulate to build a *throwaway* environment; for a *pinned, reproducible* environment use `env` / `env_manager` (or `python`) instead.
`env`, `env_manager`, `python`	Python environment selection: the reproducible alternatives to `python_version`. Use `env` + `env_manager` to point at an existing environment built by a known

tool, or python for one explicit interpreter path. env_manager is one of "system", "virtualenv", "venv", "conda", "uv", "poetry"; env is the corresponding virtualenv/conda name or path (or poetry project directory); python is an explicit interpreter path. Precedence: python > environment (env/env_manager) > python_version > default. ("virtualenv", "venv" and "uv" all point at a standard virtualenv, including one created by renv::use_python(), and behave identically.) For "virtualenv"/"venv"/"uv"/"poetry", an env that is an already-existing directory (relative to the working directory, or absolute) is resolved to an absolute path before use, so a relative venv path (e.g. ".venv", created with uv venv .venv) works as expected; otherwise reticulate::use_virtualenv() would misread a separator-less relative path as the *name* of an environment under its own virtualenv root instead of a path on disk. A bare name that does not correspond to an existing directory is passed through unchanged, so a genuine named environment (one already registered under that root) still resolves. When python or an environment is given, the selection also takes priority over an ambient RETICULATE_PYTHON environment variable (e.g. one set by the RStudio project Python config and inherited by crew workers), which reticulate would otherwise let silently override the request.

Value

The converted R object (object mode) or a character vector of normalised file paths (file mode).

See Also

run_jl_step(), tar_target_py()

Examples

```
## Not run:
# Normally invoked by tar_target_py(); shown here as a direct call.
# scripts/sum.py assigns `result`; pre.R builds `to_py <- list(x = x)`.
run_py_step(
  script = "scripts/sum.py",
  pre_script = "scripts/pre.R",
  inputs = list(x = c(1, 2, 3)),
  retrieve = "result"
)

## End(Not run)
```

---

run_rs_step *Execute a Rust step (worker behind tar_target_rs)*

Description

Compiles the `#[extendr]` functions in a Rust script with `rextendr::rust_source()`, exposing them as R functions in a fresh environment, then evaluates an R **post-script** in that environment where you call those functions and return the result. Upstream `inputs` are bound in the same environment. This is the function the target built by `tar_target_rs()` calls; it is exported so the call resolves at run time, but package users should not call it directly.

Usage

```
run_rs_step(
  script,
  post_script = NULL,
  inputs = list(),
  output = "object",
  files = NULL,
  dependencies = NULL,
  features = NULL,
  profile = NULL,
  toolchain = NULL
)
```

Arguments

`script`	Path to the Rust script containing `#[extendr]` functions (required).
`post_script`	Path to an R script evaluated after compilation. The compiled Rust functions and the named `inputs` are in scope; its last expression is the target value (object mode), or it returns a character vector of file paths (file mode). Required for object mode.
`inputs`	Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`files`	Optional character vector of file paths to return when no `post_script` is supplied in file mode.
`dependencies`, `features`, `profile`	Passed to `rextendr::rust_source()`: crate `dependencies` (named list), Cargo `features`, and build `profile` (e.g. `"dev"` or `"release"`).
`toolchain`	Optional rustup toolchain (e.g. `"stable-x86_64-pc-windows-gnu"`); sets `RUSTUP_TOOLCHAIN` for the build. Default `NULL` uses the rustup default toolchain.

Details

Unlike Python/Julia there is **no pre-script** for Rust and no live interpreter: `rust_source()` compiles a dynamic library and R calls into it with real type conversion (via extendr). A Rust toolchain

and cargo must be reachable (this function puts R, cargo, and on Windows Rtools on PATH for the build itself); on Windows use the GNU toolchain.

Value

The value of the post-script (object mode) or a character vector of normalised file paths (file mode).

See Also

tar_target_rs(), run_py_step(), run_jl_step()

Examples

```
## Not run:
# Normally invoked by tar_target_rs(); shown here as a direct call.
# scripts/square.rs defines a #[extendr] fn square(x); post.R ends on square(x).
run_rs_step(
  script = "scripts/square.rs",
  inputs = list(x = 21),
  post_script = "scripts/post.R"
)

## End(Not run)
```

---

tarpolyglot_cross *Dynamic-branching cross that compiles Rust once*

Description

A drop-in replacement for the **targets** targets::tar_target() pattern helper cross(), for use unquoted in the pattern argument of a tarpolyglot constructor. Like tarpolyglot_map(), on tar_target_rs() it compiles the extendr crate **once** in a companion <name>_rust_lib target and reuses it across every branch (instead of recompiling per branch); on tar_target_py() / tar_target_jl() it is exactly cross(). See tarpolyglot_map() for the full explanation and the compile-once mechanics.

Usage

```
tarpolyglot_cross(...)
```

Arguments

...	Upstream targets to cross (all combinations), with the same meaning as in the **targets** cross() pattern.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

tarpolyglot_map(), tar_target_rs(), tar_target_py(), tar_target_jl()

Examples

```
## Not run:
tarpolyglot::tar_target_rs(
  name = rs_grid, script = "square.rs", inputs = c(x = "a", y = "b"),
  post_script = "post.R", pattern = tarpolyglot_cross(a, b)
)

## End(Not run)
```

---

tarpolyglot_head *Dynamic-branching head that compiles Rust once*

Description

A drop-in replacement for the **targets** targets::tar_target() pattern helper head(), for use unquoted in the pattern argument of a tarpolyglot constructor. Like tarpolyglot_map(), on tar_target_rs() it compiles the extendr crate **once** in a companion <name>_rust_lib target and reuses it across the kept branches; on tar_target_py() / tar_target_jl() it is exactly head(). See tarpolyglot_map() for the full explanation and the compile-once mechanics.

Usage

```
tarpolyglot_head(..., n)
```

Arguments

...	Upstream target(s) to branch over, with the same meaning as in the **targets** head() pattern.
n	Number of branches to keep from the start, as in the **targets** head() pattern.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

tarpolyglot_map(), tar_target_rs(), tar_target_py(), tar_target_jl()

Examples

```
## Not run:
tarpolyglot::tar_target_rs(
  name = rs_first, script = "square.rs", inputs = c(x = "vals"),
  post_script = "post.R", pattern = tarpolyglot_head(vals, n = 2)
)

## End(Not run)
```

`tarpolyglot_map` *Dynamic-branching map that compiles Rust once*

Description

A drop-in replacement for the **targets** `targets::tar_target()` pattern helper `map()`, for use unquoted in the `pattern` argument of a tarpolyglot constructor (`tar_target_rs()`, `tar_target_py()`, `tar_target_jl()`, and their `_raw` forms).

Usage

```
tarpolyglot_map(...)
```

Arguments

`...`	Upstream targets to map over in parallel, exactly as in the **targets** `map()` pattern (e.g. `tarpolyglot_map(x)` or `tarpolyglot_map(x, y)`).

Details

On `tar_target_rs()` it changes how the branches are built: the extendr crate is compiled **once** in a companion target named `<name>_rust_lib`, and every branch reuses that compiled library instead of recompiling. Recompiling per branch is otherwise the default, because Rust has no live interpreter (contrast Python/Julia, whose interpreter is simply reused). With three branches this turns roughly `3 x compile` into `1 x compile + 3 x (near-instant reload)`.

On `tar_target_py()` and `tar_target_jl()` there is nothing to compile, so `tarpolyglot_map()` is exactly `map()`: it is provided so the same pipeline code can branch every language uniformly.

Use it wherever you would write `map()`: `pattern = tarpolyglot_map(x)`. The arguments are upstream target names to iterate over in parallel, with the same meaning as `targets::tar_target()`'s `map()`. This helper is only recognised inside the `pattern` argument of the tarpolyglot constructors; it is not meant to be called directly.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

tar_target_rs(), tar_target_py(), tar_target_jl()

Examples

```
## Not run:
list(
  tar_target(vals, c(10, 20, 30)),
  # Rust: compiles rs/square.rs once in `rs_branch_rust_lib`, then squares
  # each branch value reusing that compiled library.
  tarpolyglot::tar_target_rs(
    name = rs_branch,
    script = "rs/square.rs",
    inputs = c(x = "vals"),
    post_script = "R/post_square.R",
    pattern = tarpolyglot_map(vals)
  )
)

## End(Not run)
```

---

tarpolyglot_sample *Dynamic-branching sample that compiles Rust once*

Description

A drop-in replacement for the **targets** targets::tar_target() pattern helper sample(), for use unquoted in the pattern argument of a tarpolyglot constructor. Like tarpolyglot_map(), on tar_target_rs() it compiles the extendr crate **once** in a companion <name>_rust_lib target and reuses it across the sampled branches; on tar_target_py() / tar_target_jl() it is exactly sample(). See tarpolyglot_map() for the full explanation and the compile-once mechanics.

Usage

```
tarpolyglot_sample(..., n)
```

Arguments

...	Upstream target(s) to branch over, with the same meaning as in the **targets** sample() pattern.
n	Number of branches to sample at random, as in the **targets** sample() pattern.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

tarpolyglot_map(), tar_target_rs(), tar_target_py(), tar_target_jl()

Examples

```
## Not run:
tarpolyglot::tar_target_rs(
  name = rs_rand, script = "square.rs", inputs = c(x = "vals"),
  post_script = "post.R", pattern = tarpolyglot_sample(vals, n = 2)
)

## End(Not run)
```

---

tarpolyglot_slice *Dynamic-branching slice that compiles Rust once*

Description

A drop-in replacement for the **targets** targets::tar_target() pattern helper slice(), for use unquoted in the pattern argument of a tarpolyglot constructor. Like tarpolyglot_map(), on tar_target_rs() it compiles the extendr crate **once** in a companion <name>_rust_lib target and reuses it across the selected branches; on tar_target_py() / tar_target_jl() it is exactly slice(). See tarpolyglot_map() for the full explanation and the compile-once mechanics.

Usage

```
tarpolyglot_slice(..., index)
```

Arguments

...	Upstream target(s) to branch over, with the same meaning as in the **targets** slice() pattern.
index	Integer vector of branch indices to keep, as in the **targets** slice() pattern.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

tarpolyglot_map(), tar_target_rs(), tar_target_py(), tar_target_jl()

Examples

```
## Not run:
tarpolyglot::tar_target_rs(
  name = rs_some, script = "square.rs", inputs = c(x = "vals"),
  post_script = "post.R", pattern = tarpolyglot_slice(vals, index = c(1, 3))
)

## End(Not run)
```

`tarpolyglot_tail` *Dynamic-branching tail that compiles Rust once*

Description

A drop-in replacement for the **targets** `targets::tar_target()` pattern helper `tail()`, for use unquoted in the `pattern` argument of a tarpolyglot constructor. Like `tarpolyglot_map()`, on `tar_target_rs()` it compiles the extendr crate **once** in a companion `<name>_rust_lib` target and reuses it across the kept branches; on `tar_target_py()` / `tar_target_jl()` it is exactly `tail()`. See `tarpolyglot_map()` for the full explanation and the compile-once mechanics.

Usage

```
tarpolyglot_tail(..., n)
```

Arguments

`...`	Upstream target(s) to branch over, with the same meaning as in the **targets** `tail()` pattern.
`n`	Number of branches to keep from the end, as in the **targets** `tail()` pattern.

Value

This function is a marker consumed by the tarpolyglot constructors and is not meant to be evaluated on its own; calling it directly raises an error.

See Also

`tarpolyglot_map()`, `tar_target_rs()`, `tar_target_py()`, `tar_target_jl()`

Examples

```
## Not run:
tarpolyglot::tar_target_rs(
  name = rs_last, script = "square.rs", inputs = c(x = "vals"),
  post_script = "post.R", pattern = tarpolyglot_tail(vals, n = 2)
)

## End(Not run)
```

tar_code *Inline code for a tarpolyglot step*

Description

Use in place of a file-path string (or a `tar_target_path()` reference) for the `script`, `pre_script`, or `post_script` argument of the `tar_target_py()` / `tar_target_jl()` / `tar_target_rs()` constructors (and their `_raw` forms), to supply the code inline instead of pointing at a file.

Usage

```
tar_code(x)
```

Arguments

x	Inline code: an R `{...}` block, or an expression/variable/literal that yields a single character string of source code.

Details

`tar_code()` accepts two forms. An R `{...}` block is captured as inline R code and is only valid in an R slot (`pre_script` or `post_script`); passing a block as the foreign `script` is an error. Anything that evaluates to a single character string is treated as inline source code, and its language follows from the slot it fills: the foreign `script` slot is Python, Julia, or Rust source, while a `pre_script` or `post_script` slot is R source.

Multi-line strings are supported and preserved verbatim, then **dedented**: the common leading-whitespace margin shared by all lines is stripped (and leading/ trailing blank lines dropped), so code you indent to line up with the surrounding `_targets.R` still starts flush-left. This matters for Python, whose top-level indentation is significant; Julia/Rust/R are unaffected either way. For Python or regex payloads, prefer an R raw string `r"( ... )"` so backslashes and quotes need no escaping.

Because the inline code is embedded in the target's command, `targets` hashes it: editing inline code re-runs the target automatically (unlike a literal path string, which is untracked).

Value

A classed marker (`tp_inline`) recognised by the `tar_target_*` constructors.

See Also

`tar_target_path()`, `tar_target_py()`, `tar_target_jl()`, `tar_target_rs()`

Examples

```
## Not run:
# Inline R (pre/post) plus a one-line inline Python `script`:
tarpolyglot::tar_target_py(
  name        = m,
  inputs      = c(x = "prepared_x"),
  pre_script  = tar_code({ to_py <- list(x = x) }),   # inline R
  script      = tar_code("result = float(sum(x))"),   # inline Python
  post_script = tar_code({ py_get("result") })
)

# Multi-line inline Python. Indent the code to line up with `_targets.R`;
# tar_code() dedents it, so Python's own block indentation stays correct.
tarpolyglot::tar_target_py(
  name   = pos_sum,
  inputs = c(x = "prepared_x"),
  script = tar_code(r"(
    result = 0
    for v in x:
        if v > 0:
            result += v
  )")
)

## End(Not run)
```

`tar_target_jl` *Target that runs a Julia script*

Description

Non-standard-evaluation constructor mirroring `targets::tar_target()`: pass a bare `name` and unquoted `pattern`, for direct use in `_targets.R`. Delegates to `tar_target_jl_raw()`; see it and `run_jl_step()` for the full reference. The Python twin is `tar_target_py()`.

Usage

```
tar_target_jl(
  name,
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  retrieve = NULL,
  files = NULL,
  julia_version = NULL,
  julia_home = getOption("tarpolyglot.julia_home"),
```

```
  julia_project = NULL,
  julia_packages = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

`name`	Symbol, the target name (unquoted).
`script`	Path to the Julia script to run (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.
`pre_script`	Optional path to an R script run before the Julia script. See `run_jl_step()`; assign a named list `to_jl` to hand objects to Julia. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`post_script`	Optional path to an R script run after the Julia script. See `run_jl_step()`; helpers `jl_get()` and `jl_call()` are available, and its last expression (object mode) or returned paths (file mode) become the value. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`inputs`	Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`retrieve`	Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.

files — Optional character vector of file paths to return when no post_script is supplied in file mode.

julia_version — Optional Julia version to select (e.g. "1.11"), used when julia_home is not given; resolved to a juliaup-managed install. Forwarded to run_jl_step(). Default NULL uses the computer/global Julia.

julia_home, julia_project, julia_packages — Julia environment selection, forwarded to run_jl_step(). julia_home is the directory containing the julia executable (defaults to getOption("tarpolyglot.julia_home"); when unset and no julia_version, JuliaCall discovers Julia on PATH). julia_project is a Julia project environment to Pkg.activate(). julia_packages is a character vector of packages to using before the script.

pattern — Optional dynamic-branching pattern, unquoted (e.g. map(x)).

packages, library — Character vectors of R packages (and library paths) to load for the target, forwarded to targets::tar_target_raw().

deps, string — Advanced targets::tar_target_raw() arguments: extra dependency names and a string used for change detection.

format, repository, iteration — Storage/iteration settings forwarded to targets::tar_target_raw(). format defaults to "file" when output = "file", otherwise to the targets option default.

error, memory, garbage_collection, deployment, priority, resources, storage, retrieval, cue, description — Standard targets::tar_target_raw() execution/behaviour arguments, forwarded unchanged.

Value

A targets target object.

See Also

tar_target_jl_raw(), run_jl_step(), tar_target_py()

Examples

```
## Not run:
list(
  tarpolyglot::tar_target_jl(
    name = jl_sum,
    script = "scripts/sum.jl",
    inputs = c(x = "prepared_x"),
    post_script = "scripts/post.R"
  )
)

## End(Not run)
```

`tar_target_jl_raw` *Target that runs a Julia script (raw / factory form)*

Description

Character-based constructor mirroring `targets::tar_target_raw()`: `name` is a string and it is meant for use *inside targets factories*. Returns a single `targets` target whose command runs an optional R pre-script, `script` (via **JuliaCall**), and an optional R post-script, then returns a converted R object or a character vector of files. See `run_jl_step()` for the pre/post-script contract (the `to_jl` hand-off and the `jl_get` / `jl_call` helpers). For direct use in `_targets.R`, see `tar_target_jl()`. The Python twin is `tar_target_py_raw()`.

Usage

```
tar_target_jl_raw(
  name,
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  retrieve = NULL,
  files = NULL,
  julia_version = NULL,
  julia_home = getOption("tarpolyglot.julia_home"),
  julia_project = NULL,
  julia_packages = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

`name`	Character string, the target name.
`script`	Path to the Julia script to run (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.
`pre_script`	Optional path to an R script run before the Julia script. See `run_jl_step()`; assign a named list `to_jl` to hand objects to Julia. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`post_script`	Optional path to an R script run after the Julia script. See `run_jl_step()`; helpers `jl_get()` and `jl_call()` are available, and its last expression (object mode) or returned paths (file mode) become the value. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`inputs`	Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`retrieve`	Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.
`files`	Optional character vector of file paths to return when no `post_script` is supplied in file mode.
`julia_version`	Optional Julia version to select (e.g. `"1.11"`), used when `julia_home` is not given; resolved to a juliaup-managed install. Forwarded to `run_jl_step()`. Default `NULL` uses the computer/global Julia.
`julia_home`, `julia_project`, `julia_packages`	Julia environment selection, forwarded to `run_jl_step()`. `julia_home` is the directory containing the julia executable (defaults to `getOption("tarpolyglot.julia_home")`; when unset and no `julia_version`, JuliaCall discovers Julia on `PATH`). `julia_project` is a Julia project environment to `Pkg.activate()`. `julia_packages` is a character vector of packages to `using` before the script.
`pattern`	Optional **targets** dynamic-branching pattern as a language object (e.g. `quote(map(x))`), forwarded to `targets::tar_target_raw()`. The `tarpolyglot_map()` family is also accepted and behaves identically to the plain `targets` patterns here (they only differ on the Rust constructor).
`packages`, `library`	Character vectors of R packages (and library paths) to load for the target, forwarded to `targets::tar_target_raw()`.
`deps`, `string`	Advanced `targets::tar_target_raw()` arguments: extra dependency names and a string used for change detection.

`format`, `repository`, `iteration`
: Storage/iteration settings forwarded to `targets::tar_target_raw()`. `format` defaults to `"file"` when `output = "file"`, otherwise to the `targets` option default.

`error`, `memory`, `garbage_collection`, `deployment`, `priority`, `resources`, `storage`, `retrieval`, `cue`, `description`
: Standard `targets::tar_target_raw()` execution/behaviour arguments, forwarded unchanged.

Details

All `targets::tar_target_raw()` arguments are forwarded unchanged, so dynamic branching (`pattern`), storage `format`, `deployment`, `resources`, `cue`, etc. all work as usual. Upstream targets are wired in through `inputs`, which become dependencies (and are sliced under dynamic branching).

Value

A `targets` target object.

See Also

`tar_target_jl()`, `run_jl_step()`, `tar_target_py_raw()`

Examples

```
## Not run:
tarpolyglot::tar_target_jl_raw(
  name = "jl_sum",
  script = "scripts/sum.jl",
  inputs = c(x = "prepared_x"),
  post_script = "scripts/post.R",
  julia_packages = "Statistics"
)

## End(Not run)
```

`tar_target_path` *Track a script argument as a* `targets` *dependency*

Description

Use as the `script`, `pre_script`, or `post_script` argument of `tar_target_py()`/`tar_target_jl()`/`tar_target_rs()` (and their `_raw` forms) instead of a literal path string, to make that step re-run whenever the script file changes. `name` is the name of an upstream `targets` target (typically one created with `format = "file"` tracking the script file), and its *value* (the file path) is substituted in when the pipeline runs.

Usage

```
tar_target_path(name)
```

Arguments

name	Character string, the name of an upstream target whose value is the script's file path (e.g. a tar_target(..., format = "file")).

Details

This mirrors how inputs = c(x = "some_target") already wires an upstream target in by name: the target's name is given as a string, and the constructor turns it into a real dependency. A plain string passed directly as script/pre_script/post_script keeps meaning what it always has (an untracked literal path), so existing pipelines are unaffected.

Value

An object marking name for dependency-wiring by the tar_target_* constructors.

See Also

tar_target_py(), tar_target_jl(), tar_target_rs()

Examples

```
## Not run:
list(
  tar_target(mofaflex_pyscript, "python/mofaflex_fit_step.py", format = "file"),
  tar_target_py(
    name = mofaflex,
    script = tar_target_path("mofaflex_pyscript"),  # re-runs when the file changes
    pre_script = "scripts/mofaflex_pre.R",           # untracked literal path
    retrieve = "result"
  )
)

## End(Not run)
```

---

tar_target_py *Target that runs a Python script*

Description

Non-standard-evaluation constructor mirroring targets::tar_target(): pass a bare name and an unquoted pattern, for direct use in _targets.R. It quotes those and delegates to tar_target_py_raw(). See that function and run_py_step() for the full argument reference and the pre/post-script contract. The Julia twin is tar_target_jl().

Usage

```
tar_target_py(
  name,
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  retrieve = NULL,
  files = NULL,
  python_version = NULL,
  env = NULL,
  env_manager = "system",
  python = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

`name`	Symbol, the target name (unquoted).
`script`	Path to the Python script to run (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.
`pre_script`	Optional path to an R script run before the Python script. See `run_py_step()`; assign a named list `to_py` to hand objects to Python. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`post_script`	Optional path to an R script run after the Python script. See `run_py_step()`; helpers `py` and `py_get()` are available, and its last expression (object mode) or returned paths (file mode) become the value. Accepts a literal string or a `tar_target_path()` reference, as for `script`.

`inputs` Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.

`output` Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).

`retrieve` Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.

`files` Optional character vector of file paths to return when no `post_script` is supplied in file mode.

`python_version` Optional Python version to select (e.g. `"3.12"` or `">=3.11"`), used only when no environment and no explicit `python` path are given. reticulate fetches/selects it (via its uv-backed ephemeral environment) with `reticulate::py_require()`. Default `NULL` uses the computer/global default Python. Use this when you only care about the interpreter *version* and are happy for reticulate to build a *throwaway* environment; for a *pinned, reproducible* environment use `env` / `env_manager` (or `python`) instead.

`env`, `env_manager`, `python`
Python environment selection: the reproducible alternatives to `python_version`. Use `env` + `env_manager` to point at an existing environment built by a known tool, or `python` for one explicit interpreter path. `env_manager` is one of `"system"`, `"virtualenv"`, `"venv"`, `"conda"`, `"uv"`, `"poetry"`; `env` is the corresponding virtualenv/conda name or path (or poetry project directory); `python` is an explicit interpreter path. Precedence: `python` > environment (`env`/`env_manager`) > `python_version` > default. (`"virtualenv"`, `"venv"` and `"uv"` all point at a standard virtualenv, including one created by `renv::use_python()`, and behave identically.) For `"virtualenv"`/`"venv"`/`"uv"`/`"poetry"`, an `env` that is an already-existing directory (relative to the working directory, or absolute) is resolved to an absolute path before use, so a relative venv path (e.g. `".venv"`, created with `uv venv .venv`) works as expected; otherwise `reticulate::use_virtualenv()` would misread a separator-less relative path as the *name* of an environment under its own virtualenv root instead of a path on disk. A bare name that does not correspond to an existing directory is passed through unchanged, so a genuine named environment (one already registered under that root) still resolves. When `python` or an environment is given, the selection also takes priority over an ambient `RETICULATE_PYTHON` environment variable (e.g. one set by the RStudio project Python config and inherited by `crew` workers), which reticulate would otherwise let silently override the request.

`pattern` Optional dynamic-branching pattern, unquoted (e.g. `map(x)`).

`packages`, `library`
Character vectors of R packages (and library paths) to load for the target, forwarded to `targets::tar_target_raw()`.

`deps`, `string` Advanced `targets::tar_target_raw()` arguments: extra dependency names and a string used for change detection.

format, repository, iteration
: Storage/iteration settings forwarded to targets::tar_target_raw(). format defaults to "file" when output = "file", otherwise to the targets option default.

error, memory, garbage_collection, deployment, priority, resources, storage, retrieval, cue, description
: Standard targets::tar_target_raw() execution/behaviour arguments, forwarded unchanged.

Value

A targets target object.

See Also

tar_target_py_raw(), run_py_step(), tar_target_jl()

Examples

```
## Not run:
# Inside _targets.R:
list(
  tarpolyglot::tar_target_py(
    name = py_sum,
    script = "scripts/sum.py",
    inputs = c(x = "prepared_x"),
    post_script = "scripts/post.R"
  )
)

## End(Not run)
```

---

tar_target_py_raw *Target that runs a Python script (raw / factory form)*

Description

Character-based constructor mirroring targets::tar_target_raw(): name is a string and it is meant for use *inside targets factories*. Returns a single targets target whose command runs an optional R pre-script, script (via **reticulate**), and an optional R post-script, then returns a converted R object or a character vector of files. See run_py_step() for the pre/post-script contract (the to_py hand-off and the py / py_get helpers). For direct use in _targets.R, see tar_target_py(). The Julia twin is tar_target_jl_raw().

Usage

```
tar_target_py_raw(
  name,
  script,
  pre_script = NULL,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  retrieve = NULL,
  files = NULL,
  python_version = NULL,
  env = NULL,
  env_manager = "system",
  python = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

name	Character string, the target name.
script	Path to the Python script to run (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.
pre_script	Optional path to an R script run before the Python script. See `run_py_step()`; assign a named list `to_py` to hand objects to Python. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
post_script	Optional path to an R script run after the Python script. See `run_py_step()`; helpers `py` and `py_get()` are available, and its last expression (object mode) or returned paths (file mode) become the value. Accepts a literal string or a `tar_target_path()` reference, as for `script`.

`inputs` Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.

`output` Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).

`retrieve` Optional character vector of foreign-session variable names to return when no `post_script` is supplied in object mode. One name returns that object; several return a named list.

`files` Optional character vector of file paths to return when no `post_script` is supplied in file mode.

`python_version` Optional Python version to select (e.g. `"3.12"` or `">=3.11"`), used only when no environment and no explicit `python` path are given. reticulate fetches/selects it (via its uv-backed ephemeral environment) with `reticulate::py_require()`. Default `NULL` uses the computer/global default Python. Use this when you only care about the interpreter *version* and are happy for reticulate to build a *throwaway* environment; for a *pinned, reproducible* environment use `env` / `env_manager` (or `python`) instead.

`env`, `env_manager`, `python`
Python environment selection: the reproducible alternatives to `python_version`. Use `env` + `env_manager` to point at an existing environment built by a known tool, or `python` for one explicit interpreter path. `env_manager` is one of `"system"`, `"virtualenv"`, `"venv"`, `"conda"`, `"uv"`, `"poetry"`; `env` is the corresponding virtualenv/conda name or path (or poetry project directory); `python` is an explicit interpreter path. Precedence: `python` > environment (`env`/`env_manager`) > `python_version` > default. (`"virtualenv"`, `"venv"` and `"uv"` all point at a standard virtualenv, including one created by `renv::use_python()`, and behave identically.) For `"virtualenv"`/`"venv"`/`"uv"`/`"poetry"`, an `env` that is an already-existing directory (relative to the working directory, or absolute) is resolved to an absolute path before use, so a relative venv path (e.g. `".venv"`, created with `uv venv .venv`) works as expected; otherwise `reticulate::use_virtualenv()` would misread a separator-less relative path as the *name* of an environment under its own virtualenv root instead of a path on disk. A bare name that does not correspond to an existing directory is passed through unchanged, so a genuine named environment (one already registered under that root) still resolves. When `python` or an environment is given, the selection also takes priority over an ambient `RETICULATE_PYTHON` environment variable (e.g. one set by the RStudio project Python config and inherited by `crew` workers), which reticulate would otherwise let silently override the request.

`pattern` Optional **targets** dynamic-branching pattern as a language object (e.g. `quote(map(x))`), forwarded to `targets::tar_target_raw()`. The `tarpolyglot_map()` family is also accepted and behaves identically to the plain `targets` patterns here (they only differ on the Rust constructor).

`packages`, `library`
Character vectors of R packages (and library paths) to load for the target, forwarded to `targets::tar_target_raw()`.

deps, string
: Advanced targets::tar_target_raw() arguments: extra dependency names and a string used for change detection.

format, repository, iteration
: Storage/iteration settings forwarded to targets::tar_target_raw(). format defaults to "file" when output = "file", otherwise to the targets option default.

error, memory, garbage_collection, deployment, priority, resources, storage, retrieval, cue, description
: Standard targets::tar_target_raw() execution/behaviour arguments, forwarded unchanged.

Details

All targets::tar_target_raw() arguments are forwarded unchanged, so dynamic branching (pattern), storage format, deployment, resources, cue, etc. all work as usual. Upstream targets are wired in through inputs, which become dependencies (and are sliced under dynamic branching).

The interpreter/environment selection arguments (python, env, env_manager, python_version) are forwarded to run_py_step(), which documents them (they are *alternatives*: normally set only one; precedence python > env/env_manager > python_version > none). See also vignette("python") for a decision guide with examples.

Value

A targets target object.

See Also

tar_target_py(), run_py_step(), tar_target_jl_raw()

Examples

```
## Not run:
# Inside a targets factory:
tarpolyglot::tar_target_py_raw(
  name = "py_sum",
  script = "scripts/sum.py",
  inputs = c(x = "prepared_x"),
  pre_script = "scripts/pre.R",   # builds `to_py <- list(x = x)`
  post_script = "scripts/post.R"  # ends on `py$result`
)

## End(Not run)
```

tar_target_rs *Target that runs a Rust script*

Description

Non-standard-evaluation constructor mirroring `targets::tar_target()` for Rust: pass a bare `name` and unquoted `pattern`, for direct use in `_targets.R`. Compiles the `#[extendr]` functions in `script` with `rextendr::rust_source()` and calls them from the R `post_script`. Delegates to `tar_target_rs_raw()`; see it and `run_rs_step()` for the full reference. The Python/Julia twins are `tar_target_py()` / `tar_target_jl()`.

Usage

```
tar_target_rs(
  name,
  script,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  files = NULL,
  dependencies = NULL,
  features = NULL,
  profile = NULL,
  toolchain = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

`name`	Symbol, the target name (unquoted).

`script` Path to the Rust script with `#[extendr]` functions (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.

`post_script` Path to an R script run after compilation, where the compiled functions and `inputs` are in scope. Its last expression is the value (object mode); it returns file paths (file mode). Required for object mode. Accepts a literal string or a `tar_target_path()` reference, as for `script`.

`inputs` Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.

`output` Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).

`files` Optional character vector of file paths to return when no `post_script` is supplied in file mode.

`dependencies`, `features`, `profile`
Passed to `rextendr::rust_source()`: crate `dependencies` (named list), Cargo `features`, and build `profile`.

`toolchain` Optional rustup toolchain (e.g. `"stable-x86_64-pc-windows-gnu"`); sets `RUSTUP_TOOLCHAIN` for the build.

`pattern` Optional dynamic-branching pattern, unquoted (e.g. `map(x)`). Use `tarpolyglot_map(x)` to compile the crate once and reuse it across branches (see `tarpolyglot_map()`).

`packages`, `library`
Character vectors of R packages (and library paths) to load for the target, forwarded to `targets::tar_target_raw()`.

`deps`, `string` Advanced `targets::tar_target_raw()` arguments: extra dependency names and a string used for change detection.

`format`, `repository`, `iteration`
Storage/iteration settings forwarded to `targets::tar_target_raw()`. `format` defaults to `"file"` when `output = "file"`, otherwise to the `targets` option default.

`error`, `memory`, `garbage_collection`, `deployment`, `priority`, `resources`, `storage`, `retrieval`, `cue`, `description`
Standard `targets::tar_target_raw()` execution/behaviour arguments, forwarded unchanged.

Value

A `targets` target object. When `pattern` uses `tarpolyglot_map()` it is instead a list of two targets: the `<name>_rust_lib` compile target and the branched `<name>` target.

See Also

`tar_target_rs_raw()`, `tarpolyglot_map()`, `run_rs_step()`, `tar_target_py()`, `tar_target_jl()`

Examples

```
## Not run:
list(
  tarpolyglot::tar_target_rs(
    name = rs_square,
    script = "scripts/square.rs",
    inputs = c(x = "value"),
    post_script = "scripts/post.R"
  )
)

## End(Not run)
```

`tar_target_rs_raw` *Target that runs a Rust script (raw / factory form)*

Description

Character-based constructor mirroring `targets::tar_target_raw()` for Rust, for use inside targets factories. Returns a single `targets` target whose command compiles the `#[extendr]` functions in `script` with `rextendr::rust_source()` and then evaluates an R **post-script** that calls those functions (with the upstream `inputs` in scope) and returns the value. See `run_rs_step()` for the contract. For direct use in `_targets.R`, see `tar_target_rs()`.

Usage

```
tar_target_rs_raw(
  name,
  script,
  post_script = NULL,
  inputs = NULL,
  output = "object",
  files = NULL,
  dependencies = NULL,
  features = NULL,
  profile = NULL,
  toolchain = NULL,
  pattern = NULL,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  deps = NULL,
  string = NULL,
  format = NULL,
  repository = targets::tar_option_get("repository"),
  iteration = targets::tar_option_get("iteration"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
```

```
  garbage_collection = isTRUE(targets::tar_option_get("garbage_collection")),
  deployment = targets::tar_option_get("deployment"),
  priority = targets::tar_option_get("priority"),
  resources = targets::tar_option_get("resources"),
  storage = targets::tar_option_get("storage"),
  retrieval = targets::tar_option_get("retrieval"),
  cue = targets::tar_option_get("cue"),
  description = targets::tar_option_get("description")
)
```

Arguments

`name`	Character string, the target name.
`script`	Path to the Rust script with `#[extendr]` functions (required). Either a literal string (an untracked path: editing the file does not invalidate the target) or a `tar_target_path()` reference to an upstream target (typically `format = "file"`), which makes this step re-run whenever that file changes.
`post_script`	Path to an R script run after compilation, where the compiled functions and `inputs` are in scope. Its last expression is the value (object mode); it returns file paths (file mode). Required for object mode. Accepts a literal string or a `tar_target_path()` reference, as for `script`.
`inputs`	Named character vector (or list) mapping the name seen inside the step (in the R environment and, after the hand-off, in the foreign session) to the name of an upstream target, e.g. `c(x = "prepared_x")`. Each upstream target becomes a dependency of this target and is bound by that name in the step environment; under dynamic branching the per-branch slice is bound instead.
`output`	Output mode: `"object"` (default) returns a converted R object, `"file"` returns a character vector of file paths (and defaults `format` to `"file"`).
`files`	Optional character vector of file paths to return when no `post_script` is supplied in file mode.
`dependencies, features, profile`	Passed to `rextendr::rust_source()`: crate `dependencies` (named list), Cargo `features`, and build `profile`.
`toolchain`	Optional rustup toolchain (e.g. `"stable-x86_64-pc-windows-gnu"`); sets `RUSTUP_TOOLCHAIN` for the build.
`pattern`	Optional **targets** dynamic-branching pattern as a language object (e.g. `quote(map(x))`), forwarded to `targets::tar_target_raw()`. Use `quote(tarpolyglot_map(x))` to compile the crate once and reuse it across branches (see `tarpolyglot_map()`).
`packages, library`	Character vectors of R packages (and library paths) to load for the target, forwarded to `targets::tar_target_raw()`.
`deps, string`	Advanced `targets::tar_target_raw()` arguments: extra dependency names and a string used for change detection.
`format, repository, iteration`	Storage/iteration settings forwarded to `targets::tar_target_raw()`. `format` defaults to `"file"` when `output = "file"`, otherwise to the `targets` option default.

`error`, `memory`, `garbage_collection`, `deployment`, `priority`, `resources`, `storage`, `retrieval`, `cue`, `description`
: Standard `targets::tar_target_raw()` execution/behaviour arguments, forwarded unchanged.

Details

Unlike Python/Julia there is **no pre-script** for Rust: inputs are used directly in the post-script alongside the compiled functions. A Rust toolchain and `cargo` are required; on Windows use the GNU toolchain (see `vignette("rust")`).

Under dynamic branching, `pattern = map(...)` recompiles the crate in every branch (Rust has no live interpreter to reuse). Passing a tarpolyglot pattern helper instead (`tarpolyglot_map()`, `tarpolyglot_cross()`, `tarpolyglot_slice()`, `tarpolyglot_head()`, `tarpolyglot_tail()`, `tarpolyglot_sample()`) compiles the crate **once** in a companion target named `<name>_rust_lib` and reuses it across all branches; the constructor then returns *both* targets as a list. See `tarpolyglot_map()`.

Value

A `targets` target object. When `pattern` uses `tarpolyglot_map()` it is instead a list of two targets: the `<name>_rust_lib` compile target and the branched `<name>` target.

See Also

`tar_target_rs()`, `tarpolyglot_map()`, `run_rs_step()`, `tar_target_py_raw()`, `tar_target_jl_raw()`

Examples

```
## Not run:
tarpolyglot::tar_target_rs_raw(
  name = "rs_square",
  script = "scripts/square.rs",     # #[extendr] fn square(x: f64) -> f64
  inputs = c(x = "value"),
  post_script = "scripts/post.R"    # ends on square(x)
)

## End(Not run)
```

Index