Package ‘geotargets’

May 15, 2025

Title 'targets' Extensions for Geographic Spatial Formats

Version 0.3.1

Description
Provides extensions for various geographic spatial file formats, such as shape files and rasters. Currently provides support for the 'terra' geographic spatial formats. See the vignettes for worked examples, demonstrations, and explanations of how to use the various package extensions.

License MIT + file LICENSE

Encoding UTF-8

Language en-GB

RoxygenNote 7.3.2

Depends R (>= 4.1.0)

Imports targets (>= 1.8.0), rlang (>= 1.1.3), cli (>= 3.6.2), terra (>= 1.8-10), withr (>= 3.0.0), zip, lifecycle, gdalraster (>= 2.0.0)

Suggests crew (>= 0.9.2), knitr, ncmeta, rmarkdown, sf, stars, testthat (>= 3.0.0), fs, spelling

Config/testthat/edition 3

URL `https://github.com/ropensci/geotargets`, `https://docs.ropensci.org/geotargets/`

BugReports `https://github.com/ropensci/geotargets/issues`

VignetteBuilder knitr

NeedsCompilation no

Author Nicholas Tierney [aut, cre] (ORCID: <`https://orcid.org/0000-0003-1460-8722`>),
Eric Scott [aut] (ORCID: <`https://orcid.org/0000-0002-7430-7879`>),
Andrew Brown [aut] (ORCID: <`https://orcid.org/0000-0002-4565-533X`>)

Maintainer Nicholas Tierney <`nicholas.tierney@gmail.com`>

Repository CRAN

Date/Publication 2025-05-15 07:50:07 UTC

Contents

`geotargets_option_set` *Get or Set geotargets Options*

Description

Get or set behaviour for geospatial data target stores using geotargets-specific global options.

Usage

```
geotargets_option_set(
  gdal_raster_driver = NULL,
  gdal_raster_creation_options = NULL,
  gdal_raster_data_type = NULL,
  gdal_vector_driver = NULL,
  gdal_vector_creation_options = NULL,
  terra_preserve_metadata = NULL
)

geotargets_option_get(name)
```

Arguments

`gdal_raster_driver`
: character, length 1; set the driver used for raster data in target store (default: `"GTiff"`). Options for driver names can be found here: `https://gdal.org/en/stable/drivers/raster/index.html`.

`gdal_raster_creation_options`
: character; set the GDAL creation options used when writing raster files to target store (default: `""`). You may specify multiple values e.g. `c("COMPRESS=DEFLATE", "TFW=YES")`. Each GDAL driver supports a unique set of creation options. For example, with the default `"GTiff"` driver: `https://gdal.org/en/stable/drivers/raster/gtiff.html#creation-options`.

`gdal_raster_data_type`
: character; Data type for writing raster file. One of: `"INT1U"`, `"INT2U"`, `"INT4U"`, `"INT8U"`, `"INT2S"`, `"INT4S"`, `"INT8S"`, `"FLT4S"`, `"FLT8S"` (for terra), or `"Byte"`, `"UInt16"`, `"UInt32"`, `"UInt64"`, `"Int16"`, `"Int32"`, `"Int64"`, `"Float32"`, `"Float64"` (for stars).

`gdal_vector_driver`
: character, length 1; set the file type used for vector data in target store (default: `"GPKG"`).

`gdal_vector_creation_options`
: character; set the GDAL layer creation options used when writing vector files to target store (default: `"ENCODING=UTF-8"`). You may specify multiple values e.g. `c("WRITE_BBOX=YES", "COORDINATE_PRECISION=10")`. Each GDAL driver supports a unique set of creation options. For example, with the default `"GPKG"` driver: `https://gdal.org/en/stable/drivers/vector/gpkg.html#layer-creation-options`

`terra_preserve_metadata`
: character. When `"drop"` (default), any auxiliary files that would be written by `terra::writeRaster()` containing raster metadata such as units and datetimes are lost (note that this does not include layer names set with `names() <-`). When `"zip"`, these metadata are retained by archiving all written files as a zip file upon writing and unzipping them upon reading. This adds extra overhead and will slow pipelines. Also note metadata may be impacted by different versions of GDAL and different drivers. Note that you can specify this option for individual targets, e.g., inside `tar_terra_rast()` there is the option, `preserve_metadata`.

`name`
: character; option name to get.

Details

These options can also be set using `options()`. For example, `geotargets_options_set(gdal_raster_driver = "GTiff")` is equivalent to `options("geotargets.gdal.raster.driver" = "GTiff")`.

Value

Specific options, such as "gdal.raster.driver". See "Details" for more information.

Potential issues retaining metadata

If you have an issue with retaining metadata (such as units, time, etc), this could be due to the versions of GDAL and terra on your machine. We recommend exploring if this issue persists outside of geotargets. That is, try saving the file out and reading it back in using regular R code. If you find that this is an issue with geotargets, please file an issues at `https://github.com/ropensci/geotargets/issues/` and we will try and get this working for you.

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
```

```
# tar_dir() runs code from a temporary directory.
  targets::tar_dir({
    library(geotargets)
    op <- getOption("geotargets.gdal.raster.driver")
    withr::defer(options("geotargets.gdal.raster.driver" = op))
    geotargets_option_set(
      gdal_raster_driver = "COG",
      terra_preserve_metadata = "zip"
    )
    targets::tar_script({
      list(
        geotargets::tar_terra_rast(
          terra_rast_example,
          {
            new_rast <- system.file("ex/elev.tif", package = "terra") |>
              terra::rast()
            terra::units(new_rast) <- "m"
            new_rast
          }
        )
      )
    })
    targets::tar_make()
    x <- targets::tar_read(terra_rast_example)
    x
    terra::units(x)
  })
}

geotargets_option_get("gdal.raster.driver")
geotargets_option_get("gdal.raster.creation.options")
```

set_window *Copy a raster within a window*

Description

Create a new SpatRaster object as specified by a window (area of interest) over the original SpatRaster. This is a wrapper around `terra::window()` which, rather than modifying the SpatRaster in place, returns a new SpatRaster leaving the original unchanged.

Usage

```
set_window(raster, window)
```

Arguments

`raster`	a SpatRaster object.
`window`	a SpatExtent object defining the area of interest.

Value

SpatRaster

Note

While this may have general use, it was created primarily for use within `tar_terra_tiles()`.

Author(s)

Eric Scott

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- terra::rast(f)
e <- terra::ext(c(5.9, 6,49.95, 50))
r2 <- set_window(r, e)
terra::ext(r)
terra::ext(r2)
```

`tar_stars` *Create a stars* stars *Target*

Description

[Experimental]

Provides a target format for `stars` objects. Note that most or all `stars` objects work with ordinary `tar_target()` and do not necessarily *need* `geotargets` target factories the way `terra` objects do. Currently `tar_stars()` has the same limitations as `stars::write_stars()`, so use with caution.

Usage

```
tar_stars(
  name,
  command,
  pattern = NULL,
  proxy = FALSE,
  mdim = FALSE,
  ncdf = FALSE,
  driver = geotargets_option_get("gdal.raster.driver"),
  options = geotargets_option_get("gdal.raster.creation.options"),
  type = geotargets_option_get("gdal.raster.data.type"),
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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")
)

tar_stars_proxy(
  name,
  command,
  pattern = NULL,
  mdim = FALSE,
  ncdf = FALSE,
  driver = geotargets_option_get("gdal.raster.driver"),
  options = geotargets_option_get("gdal.raster.creation.options"),
  type = geotargets_option_get("gdal.raster.data.type"),
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
`command`	R code to run the target.
`pattern`	Code to define a dynamic branching pattern for a target. See `targets::tar_target()` for more information.

`proxy`	logical. Passed to `stars::read_stars()`. If `TRUE` the target will be read as an object of class `stars_proxy`. Otherwise, the object is class `stars`.
`mdim`	logical. Use the Multidimensional Raster Data Model via `stars::write_mdim()`? Default: `FALSE`. Only supported for some drivers, e.g. `"netCDF"` or `"Zarr"`.
`ncdf`	logical. Use the NetCDF library directly to read data via `stars::read_ncdf()`? Default: `FALSE`. Only supported for `driver="netCDF"`.
`driver`	character. File format expressed as GDAL driver names passed to `stars::write_stars()`. See `sf::st_drivers()`.
`options`	character. GDAL driver specific datasource creation options passed to `stars::write_stars()`.
`type`	character. character. Data type passed to `stars::write_stars()`. One of: `"Byte"`, `"UInt16"`, `"UInt32"`, `"UInt64"`, `"Int16"`, `"Int32"`, `"Int64"`, `"Float32"`, `"Float64"`.
`...`	Additional arguments not yet used.
`tidy_eval`	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
`packages`	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
`library`	Character vector of library paths to try when loading `packages`.
`repository`	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • A character string from `tar_repository_cas()` for content-addressable storage. Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
`error`	Character of length 1, what to do if the target stops and throws an error. Options: • `"stop"`: the whole pipeline stops and throws an error. • `"continue"`: the whole pipeline keeps going. • `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails.

- `"abridge"`: any currently running targets keep running, but no new targets launch after that.
- `"trim"`: all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same `tar_target()` call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. `error = "trim"` is just like `error = "abridge"`, but it allows potentially healthy regions of the dependency graph to begin running. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)

`memory` Character of length 1, memory strategy. Possible values:

- `"auto"` (default): equivalent to `memory = "transient"` in almost all cases. But to avoid superfluous reads from disk, `memory = "auto"` is equivalent to `memory = "persistent"` for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has `tar_target(name = y, command = x, pattern = map(x))`, then `tar_target(name = x, command = f(), memory = "auto")` will use persistent memory in order to avoid rereading all of `x` for every branch of `y`.
- `"transient"`: the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- `"persistent"`: the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. `format = "file"` with `repository = "aws"`), the `memory` option applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

`garbage_collection`
Logical: `TRUE` to run `base::gc()` just before the target runs, in whatever R process it is about to run (which could be a parallel worker). `FALSE` to omit garbage collection. Numeric values get converted to `FALSE`. The `garbage_collection` option in `tar_option_set()` is independent of the argument of the same name in `tar_target()`.

`deployment` Character of length 1. If `deployment` is `"main"`, then the target will run on the central controlling R process. Otherwise, if `deployment` is `"worker"` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit `https://books.ropensci.org/targets/crew.html`.

`priority` Deprecated on 2025-04-08 (`targets` version 1.10.1.9013). `targets` has moved to a more efficient scheduling algorithm (`https://github.com/ropensci/targets/issues/1458`) which cannot support priorities. The `priority` argument of `tar_target()` no longer has a reliable effect on execution order.

`resources`	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
`storage`	Character string to control when the output of the target is saved to storage. Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"worker"` (default): the worker saves/uploads the value. • `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally. • `"none"`: `targets` makes no attempt to save the result of the target to storage in the location where `targets` expects it to be. Saving to storage is the responsibility of the user. Use with caution.
`retrieval`	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`. • `"worker"`: the worker loads the target's dependencies. • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Value

target class "tar_stem" for use in a target pipeline

Note

The `iteration` argument is unavailable because it is hard-coded to `"list"`, the only option that works currently.

See Also

targets::tar_target()

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    library(geotargets)
    targets::tar_script({
      list(
        geotargets::tar_stars(
          stars_example,
          stars::read_stars(
          system.file("tif", "olinda_dem_utm25s.tif", package = "stars")
          ),
          type = "Int64"
        )
      )
    })
    targets::tar_make()
    x <- targets::tar_read(stars_example)
  })
}
```

tar_terra_rast — *Create a terra* SpatRaster *target*

Description

Provides a target format for terra::SpatRaster objects.

Usage

```
tar_terra_rast(
  name,
  command,
  pattern = NULL,
  filetype = geotargets_option_get("gdal.raster.driver"),
  gdal = geotargets_option_get("gdal.raster.creation.options"),
  datatype = geotargets_option_get("gdal.raster.data.type"),
  preserve_metadata = geotargets_option_get("terra.preserve.metadata"),
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
`command`	R code to run the target.
`pattern`	Code to define a dynamic branching pattern for a target. See `targets::tar_target()` for more information.
`filetype`	character. File format expressed as GDAL driver names passed to `terra::writeRaster()`
`gdal`	character. GDAL driver specific datasource creation options passed to `terra::writeRaster()`
`datatype`	character. Data type passed to `terra::writeRaster()`. One of: `"INT1U"`, `"INT2U"`, `"INT4U"`, `"INT8U"`, `"INT2S"`, `"INT4S"`, `"INT8S"`, `"FLT4S"`, `"FLT8S"`
`preserve_metadata`	character. When `"drop"` (default), any auxiliary files that would be written by `terra::writeRaster()` containing raster metadata such as units and datetimes are lost (note that this does not include layer names set with `names() <-`). When `"zip"`, these metadata are retained by archiving all written files as a zip file upon writing and unzipping them upon reading. This adds extra overhead and will slow pipelines. Also note metadata may be impacted by different versions of GDAL and different drivers. If you have an issue with retaining metadata for your setup, please file an issue at `https://github.com/ropensci/geotargets/issues/` and we will try and get this working for you. Also note that you can specify this option inside `geotargets_option_set()` if you want to set this for the entire pipeline.
`...`	Additional arguments passed to `terra::writeRaster()`
`tidy_eval`	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
`packages`	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
`library`	Character vector of library paths to try when loading packages.
`repository`	Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- A character string from `tar_repository_cas()` for content-addressable storage.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

`error` Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that.
- `"trim"`: all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same `tar_target()` call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. `error = "trim"` is just like `error = "abridge"`, but it allows potentially healthy regions of the dependency graph to begin running. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

`memory` Character of length 1, memory strategy. Possible values:

- `"auto"` (default): equivalent to `memory = "transient"` in almost all cases. But to avoid superfluous reads from disk, `memory = "auto"` is equivalent to `memory = "persistent"` for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has `tar_target(name = y, command = x, pattern = map(x))`, then `tar_target(name = x, command = f(), memory = "auto")` will use persistent memory in order to avoid rereading all of `x` for every branch of `y`.
- `"transient"`: the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.

- "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. format = "file" with repository = "aws"), the memory option applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical: TRUE to run base::gc() just before the target runs, in whatever R process it is about to run (which could be a parallel worker). FALSE to omit garbage collection. Numeric values get converted to FALSE. The garbage_collection option in tar_option_set() is independent of the argument of the same name in tar_target().

deployment

Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority

Deprecated on 2025-04-08 (targets version 1.10.1.9013). targets has moved to a more efficient scheduling algorithm (https://github.com/ropensci/targets/issues/1458) which cannot support priorities. The priority argument of tar_target() no longer has a reliable effect on execution order.

resources

Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage

Character string to control when the output of the target is saved to storage. Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values:

- "worker" (default): the worker saves/uploads the value.
- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "none": targets makes no attempt to save the result of the target to storage in the location where targets expects it to be. Saving to storage is the responsibility of the user. Use with caution.

retrieval

Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values:

- "auto" (default): equivalent to retrieval = "worker" in almost all cases. But to avoid unnecessary reads from disk, retrieval = "auto" is equivalent to retrieval = "main" for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has tar_target(x, command = f()), then tar_target(y, command = x, pattern = map(x), retrieval

= "auto") will use "main" retrieval in order to avoid rereading all of x for every branch of y.

- "worker": the worker loads the target's dependencies.
- "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- "none": targets makes no attempt to load its dependencies. With retrieval = "none", loading dependencies is the responsibility of the user. Use with caution.

cue	An optional object from tar_cue() to customize the rules that decide whether the target is up to date.
description	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like tar_manifest() and tar_visnetwork(), and they let you select subsets of targets for the names argument of functions like tar_make(). For example, tar_manifest(names = tar_described_as(starts_with("survival model"))) lists all the targets whose descriptions start with the character string "survival model".

Details

The terra package uses objects like terra::SpatRaster, terra::SpatVector, and terra::SpatRasterDataset (SDS), which do not contain the data directly–they contain a C++ pointer to memory where the data is stored. As a result, these objects are not portable between R sessions without special handling, which causes problems when including them in targets pipelines with targets::tar_target(). The functions, tar_terra_rast(), tar_terra_sds(), tar_terra_sprc(), tar_terra_tiles(), and tar_terra_vect() handle this issue by writing and reading the target as a geospatial file (specified by filetype) rather than saving the relevant object (e.g., SpatRaster, SpatVector, etc.), itself.

Value

target class "tar_stem" for use in a target pipeline

Note

The iteration argument is unavailable because it is hard-coded to "list", the only option that works currently.

See Also

targets::tar_target()

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    library(geotargets)
    targets::tar_script({
```

```
    list(
      geotargets::tar_terra_rast(
        terra_rast_example,
        system.file("ex/elev.tif", package = "terra") |> terra::rast()
      )
    )
  })
  targets::tar_make()
  x <- targets::tar_read(terra_rast_example)
 })
}
```

`tar_terra_sds` *Create a terra* SpatRasterDataset *target*

Description

Provides a target format for terra::SpatRasterDataset objects, which hold sub-datasets, each a `SpatRaster` that can have multiple layers.

Usage

```
tar_terra_sds(
  name,
  command,
  pattern = NULL,
  filetype = geotargets_option_get("gdal.raster.driver"),
  gdal = geotargets_option_get("gdal.raster.creation.options"),
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
command	R code to run the target.
pattern	Code to define a dynamic branching pattern for a target. See `targets::tar_target()` for more information.
filetype	character. File format expressed as GDAL driver names passed to `terra::writeRaster()`.
gdal	character. GDAL driver specific datasource creation options. passed to `terra::writeRaster()`
...	Additional arguments not yet used.
tidy_eval	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
packages	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading `packages`.
repository	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • A character string from `tar_repository_cas()` for content-addressable storage. Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
error	Character of length 1, what to do if the target stops and throws an error. Options: • `"stop"`: the whole pipeline stops and throws an error. • `"continue"`: the whole pipeline keeps going. • `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails. • `"abridge"`: any currently running targets keep running, but no new targets launch after that.

- "trim": all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same tar_target() call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. error = "trim" is just like error = "abridge", but it allows potentially healthy regions of the dependency graph to begin running. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

memory
: Character of length 1, memory strategy. Possible values:

- "auto" (default): equivalent to memory = "transient" in almost all cases. But to avoid superfluous reads from disk, memory = "auto" is equivalent to memory = "persistent" for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has tar_target(name = y, command = x, pattern = map(x)), then tar_target(name = x, command = f(), memory = "auto") will use persistent memory in order to avoid rereading all of x for every branch of y.
- "transient": the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. format = "file" with repository = "aws"), the memory option applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection
: Logical: TRUE to run base::gc() just before the target runs, in whatever R process it is about to run (which could be a parallel worker). FALSE to omit garbage collection. Numeric values get converted to FALSE. The garbage_collection option in tar_option_set() is independent of the argument of the same name in tar_target().

deployment
: Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority
: Deprecated on 2025-04-08 (targets version 1.10.1.9013). targets has moved to a more efficient scheduling algorithm (https://github.com/ropensci/targets/issues/1458) which cannot support priorities. The priority argument of tar_target() no longer has a reliable effect on execution order.

`resources`	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
`storage`	Character string to control when the output of the target is saved to storage. Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"worker"` (default): the worker saves/uploads the value. • `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally. • `"none"`: `targets` makes no attempt to save the result of the target to storage in the location where `targets` expects it to be. Saving to storage is the responsibility of the user. Use with caution.
`retrieval`	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`. • `"worker"`: the worker loads the target's dependencies. • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

The terra package uses objects like terra::SpatRaster, terra::SpatVector, and terra::SpatRasterDataset (SDS), which do not contain the data directly–they contain a C++ pointer to memory where the data is stored. As a result, these objects are not portable between R sessions without special handling, which causes problems when including them in `targets` pipelines with `targets::tar_target()`. The functions, `tar_terra_rast()`, `tar_terra_sds()`, `tar_terra_sprc()`, `tar_terra_tiles()`, and `tar_terra_vect()` handle this issue by writing and reading the target as a geospatial file (specified by `filetype`) rather than saving the relevant object (e.g., `SpatRaster`, `SpatVector`, etc.), itself.

Value

target class "tar_stem" for use in a target pipeline

Note

The `iteration` argument is unavailable because it is hard-coded to `"list"`, the only option that works currently.

Author(s)

Andrew Gene Brown

Nicholas Tierney

Eric R. Scott

See Also

`targets::tar_target_raw()`, `tar_terra_sprc()`

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    targets::tar_script({
      library(geotargets)
      elev_scale <- function(z = 1) {
        terra::rast(system.file("ex", "elev.tif", package = "terra")) * z
      }
      list(
        tar_terra_sds(
          raster_elevs,
          # two rasters, one unaltered, one scaled by factor of 2
          command = terra::sds(list(
            elev_scale(1),
            elev_scale(2)
          ))
        )
      )
    })
    targets::tar_make()
    targets::tar_read(raster_elevs)
  })
}
```

`tar_terra_sprc` *Create a terra* SpatRasterCollection *target*

Description

Provides a target format for terra::SpatRasterCollection objects, which have no restriction in the extent or other geometric parameters.

Usage

```
tar_terra_sprc(
  name,
  command,
  pattern = NULL,
  filetype = geotargets_option_get("gdal.raster.driver"),
  gdal = geotargets_option_get("gdal.raster.creation.options"),
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
`command`	R code to run the target.
`pattern`	Code to define a dynamic branching pattern for a target. See `targets::tar_target()` for more information.
`filetype`	character. File format expressed as GDAL driver names passed to `terra::writeRaster()`.
`gdal`	character. GDAL driver specific datasource creation options. passed to `terra::writeRaster()`
`...`	Additional arguments not yet used.

`tidy_eval`	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
`packages`	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
`library`	Character vector of library paths to try when loading `packages`.
`repository`	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • A character string from `tar_repository_cas()` for content-addressable storage. Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.
`error`	Character of length 1, what to do if the target stops and throws an error. Options: • `"stop"`: the whole pipeline stops and throws an error. • `"continue"`: the whole pipeline keeps going. • `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails. • `"abridge"`: any currently running targets keep running, but no new targets launch after that. • `"trim"`: all currently running targets stay running. A queued target is allowed to start if: 1. It is not downstream of the error, and 2. It is not a sibling branch from the same `tar_target()` call (if the error happened in a dynamic branch). The idea is to avoid starting any new work that the immediate error impacts. `error = "trim"` is just like `error = "abridge"`, but it allows potentially healthy regions of the dependency graph to begin running. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)
`memory`	Character of length 1, memory strategy. Possible values:

- "auto" (default): equivalent to memory = "transient" in almost all cases. But to avoid superfluous reads from disk, memory = "auto" is equivalent to memory = "persistent" for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has tar_target(name = y, command = x, pattern = map(x)), then tar_target(name = x, command = f(), memory = "auto") will use persistent memory in order to avoid rereading all of x for every branch of y.
- "transient": the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. format = "file" with repository = "aws"), the memory option applies to the temporary local copy of the file: "persistent" means it remains until the end of the pipeline and is then deleted, and "transient" means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

garbage_collection

Logical: TRUE to run base::gc() just before the target runs, in whatever R process it is about to run (which could be a parallel worker). FALSE to omit garbage collection. Numeric values get converted to FALSE. The garbage_collection option in tar_option_set() is independent of the argument of the same name in tar_target().

deployment

Character of length 1. If deployment is "main", then the target will run on the central controlling R process. Otherwise, if deployment is "worker" and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in targets, please visit https://books.ropensci.org/targets/crew.html.

priority

Deprecated on 2025-04-08 (targets version 1.10.1.9013). targets has moved to a more efficient scheduling algorithm (https://github.com/ropensci/targets/issues/1458) which cannot support priorities. The priority argument of tar_target() no longer has a reliable effect on execution order.

resources

Object returned by tar_resources() with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See tar_resources() for details.

storage

Character string to control when the output of the target is saved to storage. Only relevant when using targets with parallel workers (https://books.ropensci.org/targets/crew.html). Must be one of the following values:

- "worker" (default): the worker saves/uploads the value.
- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "none": targets makes no attempt to save the result of the target to storage in the location where targets expects it to be. Saving to storage is the responsibility of the user. Use with caution.

`retrieval`	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`. • `"worker"`: the worker loads the target's dependencies. • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

The terra package uses objects like terra::SpatRaster, terra::SpatVector, and terra::SpatRasterDataset (SDS), which do not contain the data directly–they contain a C++ pointer to memory where the data is stored. As a result, these objects are not portable between R sessions without special handling, which causes problems when including them in `targets` pipelines with `targets::tar_target()`. The functions, `tar_terra_rast()`, `tar_terra_sds()`, `tar_terra_sprc()`, `tar_terra_tiles()`, and `tar_terra_vect()` handle this issue by writing and reading the target as a geospatial file (specified by `filetype`) rather than saving the relevant object (e.g., `SpatRaster`, `SpatVector`, etc.), itself.

Value

target class "tar_stem" for use in a target pipeline

Note

The `iteration` argument is unavailable because it is hard-coded to `"list"`, the only option that works currently.

Author(s)

Andrew Gene Brown

Nicholas Tierney

See Also

`targets::tar_target_raw()`

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    targets::tar_script({
      library(geotargets)
      elev_scale <- function(z = 1, projection = "EPSG:4326") {
        terra::project(
          terra::rast(system.file("ex", "elev.tif", package = "terra")) * z,
          projection
        )
      }
      list(
        tar_terra_sprc(
          raster_elevs,
          # two rasters, one unaltered, one scaled by factor of 2 and
          # reprojected to interrupted good homolosine
          command = terra::sprc(list(
            elev_scale(1),
            elev_scale(2, "+proj=igh")
          ))
        )
      )
    })
    targets::tar_make()
    targets::tar_read(raster_elevs)
  })
}
```

`tar_terra_tiles` *Split a raster into tiles that can be iterated over with dynamic branching*

Description

Creates two targets, a list of extents defining tiles and a downstream pattern that maps over these extents to create a list of `SpatRaster` objects that can be used with dynamic branching.

Usage

```
tar_terra_tiles(
  name,
  raster,
  tile_fun,
  filetype = geotargets_option_get("gdal.raster.driver"),
  gdal = geotargets_option_get("gdal.raster.creation.options"),
  ...,
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
`raster`	a `SpatRaster` object to be split into tiles.
`tile_fun`	a helper function that returns a list of numeric vectors such as `tile_grid()`, `tile_n()` or tile_blocksize specified in one of the following ways: • A named function, e.g. `tile_blocksize` or `"tile_blocksize"`. • An anonymous function, e.g. `\(x) tile_grid(x, nrow = 2, ncol = 2)`.
`filetype`	character. File format expressed as GDAL driver names passed to `terra::makeTiles()`.
`gdal`	character. GDAL driver specific datasource creation options passed to `terra::makeTiles()`.
`...`	additional arguments not yet used.
`packages`	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
`library`	Character vector of library paths to try when loading packages.
`repository`	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`,

but versioning capabilities may be lost in doing so. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.

- "gcp": Google Cloud Platform storage bucket. See the cloud storage section of https://books.ropensci.org/targets/data.html for details for instructions.
- A character string from tar_repository_cas() for content-addressable storage.

Note: if repository is not "local" and format is "file" then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

error Character of length 1, what to do if the target stops and throws an error. Options:

- "stop": the whole pipeline stops and throws an error.
- "continue": the whole pipeline keeps going.
- "null": The errored target continues and returns NULL. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of targets version 1.8.0.9011, a value of NULL is given to upstream dependencies with error = "null" if loading fails.
- "abridge": any currently running targets keep running, but no new targets launch after that.
- "trim": all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same tar_target() call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. error = "trim" is just like error = "abridge", but it allows potentially healthy regions of the dependency graph to begin running. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)

memory Character of length 1, memory strategy. Possible values:

- "auto" (default): equivalent to memory = "transient" in almost all cases. But to avoid superfluous reads from disk, memory = "auto" is equivalent to memory = "persistent" for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has tar_target(name = y, command = x, pattern = map(x)), then tar_target(name = x, command = f(), memory = "auto") will use persistent memory in order to avoid rereading all of x for every branch of y.
- "transient": the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- "persistent": the target stays in memory until the end of the pipeline (unless storage is "worker", in which case targets unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. `format = "file"` with `repository = "aws"`), the `memory` option applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

`garbage_collection`
: Logical: `TRUE` to run `base::gc()` just before the target runs, in whatever R process it is about to run (which could be a parallel worker). `FALSE` to omit garbage collection. Numeric values get converted to `FALSE`. The `garbage_collection` option in `tar_option_set()` is independent of the argument of the same name in `tar_target()`.

`deployment`
: Character of length 1. If `deployment` is `"main"`, then the target will run on the central controlling R process. Otherwise, if `deployment` is `"worker"` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit `https://books.ropensci.org/targets/crew.html`.

`priority`
: Deprecated on 2025-04-08 (`targets` version 1.10.1.9013). `targets` has moved to a more efficient scheduling algorithm (`https://github.com/ropensci/targets/issues/1458`) which cannot support priorities. The `priority` argument of `tar_target()` no longer has a reliable effect on execution order.

`resources`
: Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.

`storage`
: Character string to control when the output of the target is saved to storage. Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values:

- `"worker"` (default): the worker saves/uploads the value.
- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"none"`: `targets` makes no attempt to save the result of the target to storage in the location where `targets` expects it to be. Saving to storage is the responsibility of the user. Use with caution.

`retrieval`
: Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values:

- `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`.
- `"worker"`: the worker loads the target's dependencies.

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.

`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

When a raster is too large or too high resolution to work on in-memory, one possible solution is to iterate over tiles. Raster tiles can then be operated on one at a time, or possibly in parallel if resources are available, and then the results can be aggregated. A natural way to do this in the context of a `targets` pipeline is to split the raster into multiple raster targets with dynamic branching so that downstream targets can be applied to each branch of the upstream target with the `pattern` argument to `tar_terra_rast()` or `tar_target()`. `tar_terra_tiles()` facilitates creation of such a dynamically branched target. This workflow isn't appropriate for operations that aggregate spatially, only pixel-wise operations (possibly aggregating across multiple layers).

This target factory is useful when a raster is too large or too high resolution to work on in-memory. It can instead be split into tiles that can be iterated over using dynamic branching.

Value

a list of two targets: an upstream target that creates a list of extents and a downstream pattern that maps over these extents to create a list of SpatRaster objects.

Note

The `iteration` argument is unavailable because it is hard-coded to `"list"`, the only option that works currently.

When using the `tile_blocksize()` helper function, you may need to set `memory = "transient"` on the upstream target provided to the `raster` argument of `tar_terra_tiles()`. More details are in the help file for `tile_blocksize()`.

Author(s)

Eric Scott

See Also

`tile_n()`, `tile_grid()`, `tile_blocksize()`, `tar_terra_rast()`, `tar_terra_vrt()`

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({
    targets::tar_script({
        library(targets)
        library(geotargets)
        library(terra)
        list(
            tar_target(
                my_file,
                system.file("ex/elev.tif", package="terra"),
                format = "file"
            ),
            tar_terra_rast(
                my_map,
                terra::rast(my_file)
            ),
            tar_terra_tiles(
                name = rast_split,
                raster = my_map,
                tile_fun = \(x) tile_grid(x, ncol = 2, nrow = 2)
            )
        )
    })
    targets::tar_manifest()
  })
}
```

`tar_terra_vect` *Create a terra* SpatVector *target*

Description

Provides a target format for terra::SpatVector objects.

Usage

```
tar_terra_vect(
  name,
  command,
  pattern = NULL,
  filetype = geotargets_option_get("gdal.vector.driver"),
  gdal = geotargets_option_get("gdal.vector.creation.options"),
  ...,
  packages = targets::tar_option_get("packages"),
  tidy_eval = targets::tar_option_get("tidy_eval"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. A target name must be a valid name for a symbol in R, and it must not start with a dot. See `targets::tar_target()` for more information.
`command`	R code to run the target.
`pattern`	Code to define a dynamic branching pattern for a target. See `targets::tar_target()` for more information.
`filetype`	character. File format expressed as GDAL driver names passed to `terra::writeVector()`. See 'Note' for more details.
`gdal`	character. GDAL driver specific datasource creation options passed to `terra::writeVector()`.
`...`	Additional arguments passed to `terra::writeVector()`
`packages`	Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.
`tidy_eval`	Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.
`library`	Character vector of library paths to try when loading packages.
`repository`	Character of length 1, remote repository for target storage. Choices: • `"local"`: file system of the local machine. • `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions. • A character string from `tar_repository_cas()` for content-addressable storage.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

`error` Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that.
- `"trim"`: all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same `tar_target()` call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. `error = "trim"` is just like `error = "abridge"`, but it allows potentially healthy regions of the dependency graph to begin running. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)

`memory` Character of length 1, memory strategy. Possible values:

- `"auto"` (default): equivalent to `memory = "transient"` in almost all cases. But to avoid superfluous reads from disk, `memory = "auto"` is equivalent to `memory = "persistent"` for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has `tar_target(name = y, command = x, pattern = map(x))`, then `tar_target(name = x, command = f(), memory = "auto")` will use persistent memory in order to avoid rereading all of `x` for every branch of `y`.
- `"transient"`: the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- `"persistent"`: the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. `format = "file"` with `repository = "aws"`), the `memory` option applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

`garbage_collection`

Logical: `TRUE` to run `base::gc()` just before the target runs, in whatever R process it is about to run (which could be a parallel worker). `FALSE` to omit garbage collection. Numeric values get converted to `FALSE`. The `garbage_collection` option in `tar_option_set()` is independent of the argument of the same name in `tar_target()`.

`deployment`	Character of length 1. If `deployment` is `"main"`, then the target will run on the central controlling R process. Otherwise, if `deployment` is `"worker"` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit `https://books.ropensci.org/targets/crew.html`.
`priority`	Deprecated on 2025-04-08 (`targets` version 1.10.1.9013). `targets` has moved to a more efficient scheduling algorithm (`https://github.com/ropensci/targets/issues/1458`) which cannot support priorities. The `priority` argument of `tar_target()` no longer has a reliable effect on execution order.
`resources`	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
`storage`	Character string to control when the output of the target is saved to storage. Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"worker"` (default): the worker saves/uploads the value. • `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally. • `"none"`: `targets` makes no attempt to save the result of the target to storage in the location where `targets` expects it to be. Saving to storage is the responsibility of the user. Use with caution.
`retrieval`	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`. • `"worker"`: the worker loads the target's dependencies. • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.

`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

The terra package uses objects like terra::SpatRaster, terra::SpatVector, and terra::SpatRasterDataset (SDS), which do not contain the data directly–they contain a C++ pointer to memory where the data is stored. As a result, these objects are not portable between R sessions without special handling, which causes problems when including them in `targets` pipelines with `targets::tar_target()`. The functions, `tar_terra_rast()`, `tar_terra_sds()`, `tar_terra_sprc()`, `tar_terra_tiles()`, and `tar_terra_vect()` handle this issue by writing and reading the target as a geospatial file (specified by `filetype`) rather than saving the relevant object (e.g., `SpatRaster`, `SpatVector`, etc.), itself.

Value

target class "tar_stem" for use in a target pipeline

Note

The `iteration` argument is unavailable because it is hard-coded to `"list"`, the only option that works currently.

Although you may pass any supported GDAL vector driver to the `filetype` argument, not all formats are guaranteed to work with `geotargets`. At the moment, we have tested `GPKG`, `GeoJSON` and `ESRI Shapefile` which all appear to work generally.

Examples

```
# For CRAN. Ensures these examples run under certain conditions.
# To run this locally, run the code inside this if statement
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    targets::tar_script({
      lux_area <- function(projection = "EPSG:4326") {
        terra::project(
          terra::vect(system.file("ex", "lux.shp",
            package = "terra"
          )),
          projection
        )
      }
      list(
        geotargets::tar_terra_vect(
          terra_vect_example,
          lux_area()
        )
      )
    })
    targets::tar_make()
    x <- targets::tar_read(terra_vect_example)
  })
}
```

tar_terra_vrt *Create a GDAL Virtual Dataset (VRT) with terra*

Description

Provides a target format for terra::SpatRaster, terra::SpatRasterDataset, and terra::SpatRasterCollection objects representing a GDAL Virtual Dataset (VRT).

Usage

```
tar_terra_vrt(
  name,
  command,
  pattern = NULL,
  ...,
  tidy_eval = targets::tar_option_get("tidy_eval"),
  packages = targets::tar_option_get("packages"),
  library = targets::tar_option_get("library"),
  repository = targets::tar_option_get("repository"),
  error = targets::tar_option_get("error"),
  memory = targets::tar_option_get("memory"),
  garbage_collection = 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, name of the target. In `tar_target()`, name is an unevaluated symbol, e.g. `tar_target(name = data)`. In `tar_target_raw()`, name is a character string, e.g. `tar_target_raw(name = "data")`. A target name must be a valid name for a symbol in R, and it must not start with a dot. Subsequent targets can refer to this name symbolically to induce a dependency relationship: e.g. `tar_target(downstream_target, f(upstream_target))` is a target named `downstream_target` which depends on a target `upstream_target` and a function `f()`.

In most cases, The target name is the name of its local data file in storage. Some file systems are not case sensitive, which means converting a name to a different case may overwrite a different target. Please ensure all target names have unique names when converted to lower case.

In addition, a target's name determines its random number generator seed. In this way, each target runs with a reproducible seed so someone else running the same pipeline should get the same results, and no two targets in the same pipeline share the same seed. (Even dynamic branches have different names and thus different seeds.) You can recover the seed of a completed target with `tar_meta(your_target, seed)` and run `tar_seed_set()` on the result to locally recreate the target's initial RNG state.

`command` R code to run the target. In `tar_target()`, `command` is an unevaluated expression, e.g. `tar_target(command = data)`. In `tar_target_raw()`, `command` is an evaluated expression, e.g. `tar_target_raw(command = quote(data))`.

`pattern` Code to define a dynamic branching branching for a target. In `tar_target()`, `pattern` is an unevaluated expression, e.g. `tar_target(pattern = map(data))`. In `tar_target_raw()`, `command` is an evaluated expression, e.g. `tar_target_raw(pattern = quote(map(data)))`.

To demonstrate dynamic branching patterns, suppose we have a pipeline with numeric vector targets `x` and `y`. Then, `tar_target(z, x + y, pattern = map(x, y))` implicitly defines branches of `z` that each compute `x[1] + y[1]`, `x[2] + y[2]`, and so on. See the user manual for details.

`...` Additional arguments passed to `terra::vrt()`

`tidy_eval` Logical, whether to enable tidy evaluation when interpreting `command` and `pattern`. If `TRUE`, you can use the "bang-bang" operator `!!` to programmatically insert the values of global objects.

`packages` Character vector of packages to load right before the target runs or the output data is reloaded for downstream targets. Use `tar_option_set()` to set packages globally for all subsequent targets you define.

`library` Character vector of library paths to try when loading `packages`.

`repository` Character of length 1, remote repository for target storage. Choices:

- `"local"`: file system of the local machine.
- `"aws"`: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the `endpoint` argument of `tar_resources_aws()`, but versioning capabilities may be lost in doing so. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.
- `"gcp"`: Google Cloud Platform storage bucket. See the cloud storage section of `https://books.ropensci.org/targets/data.html` for details for instructions.
- A character string from `tar_repository_cas()` for content-addressable storage.

Note: if `repository` is not `"local"` and `format` is `"file"` then the target should create a single output file. That output file is uploaded to the cloud and tracked for changes where it exists in the cloud. The local file is deleted after the target runs.

`error` Character of length 1, what to do if the target stops and throws an error. Options:

- `"stop"`: the whole pipeline stops and throws an error.
- `"continue"`: the whole pipeline keeps going.
- `"null"`: The errored target continues and returns `NULL`. The data hash is deliberately wrong so the target is not up to date for the next run of the pipeline. In addition, as of `targets` version 1.8.0.9011, a value of `NULL` is given to upstream dependencies with `error = "null"` if loading fails.
- `"abridge"`: any currently running targets keep running, but no new targets launch after that.
- `"trim"`: all currently running targets stay running. A queued target is allowed to start if:
  1. It is not downstream of the error, and
  2. It is not a sibling branch from the same `tar_target()` call (if the error happened in a dynamic branch).

  The idea is to avoid starting any new work that the immediate error impacts. `error = "trim"` is just like `error = "abridge"`, but it allows potentially healthy regions of the dependency graph to begin running. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)

`memory` Character of length 1, memory strategy. Possible values:

- `"auto"` (default): equivalent to `memory = "transient"` in almost all cases. But to avoid superfluous reads from disk, `memory = "auto"` is equivalent to `memory = "persistent"` for for non-dynamically-branched targets that other targets dynamically branch over. For example: if your pipeline has `tar_target(name = y, command = x, pattern = map(x))`, then `tar_target(name = x, command = f(), memory = "auto")` will use persistent memory in order to avoid rereading all of `x` for every branch of `y`.
- `"transient"`: the target gets unloaded after every new target completes. Either way, the target gets automatically loaded into memory whenever another target needs the value.
- `"persistent"`: the target stays in memory until the end of the pipeline (unless `storage` is `"worker"`, in which case `targets` unloads the value from memory right after storing it in order to avoid sending copious data over a network).

For cloud-based file targets (e.g. `format = "file"` with `repository = "aws"`), the `memory` option applies to the temporary local copy of the file: `"persistent"` means it remains until the end of the pipeline and is then deleted, and `"transient"` means it gets deleted as soon as possible. The former conserves bandwidth, and the latter conserves local storage.

`garbage_collection`

Logical: `TRUE` to run `base::gc()` just before the target runs, in whatever R process it is about to run (which could be a parallel worker). `FALSE` to omit garbage collection. Numeric values get converted to `FALSE`. The `garbage_collection` option in `tar_option_set()` is independent of the argument of the same name in `tar_target()`.

`deployment`	Character of length 1. If `deployment` is `"main"`, then the target will run on the central controlling R process. Otherwise, if `deployment` is `"worker"` and you set up the pipeline with distributed/parallel computing, then the target runs on a parallel worker. For more on distributed/parallel computing in `targets`, please visit `https://books.ropensci.org/targets/crew.html`.
`priority`	Deprecated on 2025-04-08 (`targets` version 1.10.1.9013). `targets` has moved to a more efficient scheduling algorithm (`https://github.com/ropensci/targets/issues/1458`) which cannot support priorities. The `priority` argument of `tar_target()` no longer has a reliable effect on execution order.
`resources`	Object returned by `tar_resources()` with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of `targets`. See `tar_resources()` for details.
`storage`	Character string to control when the output of the target is saved to storage. Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"worker"` (default): the worker saves/uploads the value. • `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally. • `"none"`: `targets` makes no attempt to save the result of the target to storage in the location where `targets` expects it to be. Saving to storage is the responsibility of the user. Use with caution.
`retrieval`	Character string to control when the current target loads its dependencies into memory before running. (Here, a "dependency" is another target upstream that the current one depends on.) Only relevant when using `targets` with parallel workers (`https://books.ropensci.org/targets/crew.html`). Must be one of the following values: • `"auto"` (default): equivalent to `retrieval = "worker"` in almost all cases. But to avoid unnecessary reads from disk, `retrieval = "auto"` is equivalent to `retrieval = "main"` for dynamic branches that branch over non-dynamic targets. For example: if your pipeline has `tar_target(x, command = f())`, then `tar_target(y, command = x, pattern = map(x), retrieval = "auto")` will use `"main"` retrieval in order to avoid rereading all of `x` for every branch of `y`. • `"worker"`: the worker loads the target's dependencies. • `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • `"none"`: `targets` makes no attempt to load its dependencies. With `retrieval = "none"`, loading dependencies is the responsibility of the user. Use with caution.
`cue`	An optional object from `tar_cue()` to customize the rules that decide whether the target is up to date.
`description`	Character of length 1, a custom free-form human-readable text description of the target. Descriptions appear as target labels in functions like `tar_manifest()` and `tar_visnetwork()`, and they let you select subsets of targets for the `names` argument of functions like `tar_make()`. For example, `tar_manifest(names = tar_described_as(starts_with("survival model")))` lists all the targets whose descriptions start with the character string `"survival model"`.

Details

`tar_terra_vrt()` accepts SpatRaster, SpatRasterDataset, or SpatRasterCollection objects as input, and returns a SpatRaster referencing a GDAL Virtual Dataset file (.vrt). The .vrt file format uses XML and describes the layers and tiles that comprise a virtual raster data source. To use a list of SpatRaster of varying extent, such as output from `tar_terra_tiles()`, or a character vector of paths, wrap the tile result in a call to `terra::sprc()` to create a SpatRasterCollection.

Value

target class "tar_stem" for use in a target pipeline

See Also

`tar_terra_tiles()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
 targets::tar_dir({ # tar_dir() runs code from a temporary directory.
   targets::tar_script({
     list(
       geotargets::tar_terra_vrt(
         terra_rast_example,
         terra::rast(system.file("ex/elev.tif", package = "terra"))
       )
     )
   })
   targets::tar_make()
   x <- targets::tar_read(terra_rast_example)
 })

targets::tar_dir({
    targets::tar_script({
        library(targets)
        library(geotargets)
        list(
            tar_terra_rast(r, terra::rast(
                system.file("ex", "elev.tif", package = "terra")
            )),
            tar_terra_rast(r2, r * 2),
            tar_terra_tiles(rt, c(r, r2), function(x)
                tile_grid(x, ncol = 2, nrow = 2)),
            tar_terra_vrt(r3, terra::sprc(rt))
        )
    })
})

}
```

tile_grid *Helper functions to create tiles*

Description

Wrappers around `terra::getTileExtents()` that return a list of named numeric vectors describing the extents of tiles rather than `SpatExtent` objects. While these may have general use, they are intended primarily for supplying to the `tile_fun` argument of `tar_terra_tiles()`.

Usage

```
tile_grid(raster, ncol, nrow)

tile_blocksize(raster, n_blocks_row = 1, n_blocks_col = 1)

tile_n(raster, n)
```

Arguments

`raster`	a SpatRaster object.
`ncol`	integer; number of columns to split the SpatRaster into.
`nrow`	integer; number of rows to split the SpatRaster into.
`n_blocks_row`	integer; multiple of blocksize to include in each tile vertically.
`n_blocks_col`	integer; multiple of blocksize to include in each tile horizontally.
`n`	integer; total number of tiles to split the SpatRaster into.

Details

`tile_blocksize()` creates extents using the raster's native block size (see `terra::fileBlocksize()`), which should be more memory efficient. Create tiles with multiples of the raster's blocksize with `n_blocks_row` and `n_blocks_col`. We strongly suggest the user explore how many tiles are created by `tile_blocksize()` before creating a dynamically branched target using this helper. Note that block size is a property of *files* and does not apply to in-memory `SpatRasters`. Therefore, if you want to use this helper in `tar_terra_tiles()` you may need to ensure the upstream target provided to the `raster` argument is not in memory by setting `memory = "transient"`.

`tile_grid()` allows specification of a number of rows and columns to split the raster into. E.g. nrow = 2 and ncol = 2 would create 4 tiles (because it specifies a 2x2 matrix, which has 4 elements).

`tile_n()` creates (about) `n` tiles and prints the number of rows, columns, and total tiles created.

Value

list of named numeric vectors with xmin, xmax, ymin, and ymax values that can be coerced to SpatExtent objects with `terra::ext()`.

Author(s)

Eric Scott

Examples

```
f <- system.file("ex/elev.tif", package="terra")
r <- terra::rast(f)
tile_grid(r, ncol = 2, nrow = 2)
tile_blocksize(r)
tile_n(r, 8)


#Example usage with tar_terra_tiles
list(
    tar_terra_rast(
        my_map,
        terra::rast(system.file("ex/logo.tif", package = "terra"))
    ),
    tar_terra_tiles(
        name = rast_split,
        raster = my_map,
        tile_fun = tile_blocksize,
        description = "Each tile is 1 block"
    ),
    tar_terra_tiles(
        name = rast_split_2blocks,
        raster = my_map,
        tile_fun = \(x) tile_blocksize(
          x,
          n_blocks_row = 2,
          n_blocks_col = 1
          ),
        description = "Each tile is 2 blocks tall, 1 block wide"
    ),
    tar_terra_tiles(
        name = rast_split_grid,
        raster = my_map,
        tile_fun = \(x) tile_grid(x, ncol = 2, nrow = 2),
        description = "Split into 4 tiles in a 2x2 grid"
    ),
    tar_terra_tiles(
        name = rast_split_n,
        raster = my_map,
        tile_fun = \(x) tile_n(x, n = 6),
        description = "Split into 6 tiles"
    )
)
```

Index