Package: geotargets (via r-universe)

July 17, 2024

Title 'Targets' Extensions for Geospatial Formats

Version 0.1.0.9000

Description Provides extensions for various geospatial file formats, such as shapefiles and rasters. Currently provides support for the 'terra' geospatial 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

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.1

Imports targets (>= 1.7.0), rlang (>= 1.1.3), cli (>= 3.6.2)

Suggests crew (>= 0.9.2), ncmeta, sf, stars, terra (>= 1.7.71), testthat (>= 3.0.0), withr (>= 3.0.0)

Config/testthat/edition 3

URL https://github.com/njtierney/geotargets, https://njtierney.github.io/geotargets/

BugReports https://github.com/njtierney/geotargets/issues

Repository https://njtierney.r-universe.dev

RemoteUrl https://github.com/njtierney/geotargets

RemoteRef HEAD

RemoteSha 76285d670cb8c176415835436616ea2bbd76003f

Contents

`create_tile_exts` *Create extents for raster tiles*

Description

A wrapper around `terra::getTileExtents()` for creating a tile specification. Rather than having to supply a template, one is created from the original raster extent and CRS using `ncol` and `nrow`. The output is a list of named numeric vectors that can be coerced to SpatExtent objects with `terra::ext()`.

Usage

```
create_tile_exts(raster, ncol, nrow)
```

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

Value

list of named numeric vectors with xmin, xmax, ymin, and ymax values

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)
r_tiles <- create_tile_exts(r, ncol = 2, nrow = 2)
r_tiles
```

geotargets_option_set *Get or Set geotargets Options*

Description

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

Usage

```
geotargets_option_set(
  gdal_raster_driver = NULL,
  gdal_raster_creation_options = NULL,
  gdal_vector_driver = NULL,
  gdal_vector_creation_options = 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/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/drivers/raster/gtiff.html#creation-options
gdal_vector_driver	character, length 1; set the file type used for vector data in target store (default: "GeoJSON").
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 "GeoJSON" driver: https://gdal.org/drivers/vector/geojson.html#layer-creation-options
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.

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
 targets::tar_dir({ # tar_dir() runs code from a temporary directory.
   library(geotargets)
  op <- getOption("geotargets.gdal.raster.driver")
  withr::defer(options("geotargets.gdal.raster.driver" = op))
  geotargets_option_set(gdal_raster_driver = "COG")
   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)
 })
}

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

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

Provides a target format for stars objects.

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"),
  ...,
  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"),
  iteration = targets::tar_option_get("iteration"),
  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"),
  ...,
  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"),
  iteration = targets::tar_option_get("iteration"),
  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. 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 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.
pattern	Language to define branching for a target. For example, in a pipeline with numeric vector targets `x` and `y`, `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.

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()

... — 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.

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.

iteration — Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of tar_target() must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().

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.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "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.

memory
: Character of length 1, memory strategy. If "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). If "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. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy 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, whether to run base::gc() just before the target runs.

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
: Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

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 of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

  - "main": the target's return value is sent back to the host machine and saved/uploaded locally.
  - "worker": the worker saves/uploads the value.
  - "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
    If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval` Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`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"`.

See Also

`targets::tar_target_raw()`

Examples

```
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"))
       )
     )
   })
   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"),
  ...,
  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"),
  iteration = targets::tar_option_get("iteration"),
  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. 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 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.

`pattern` Language to define branching for a target. For example, in a pipeline with numeric vector targets `x` and `y`, `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.

`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.

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.

`iteration` Character of length 1, name of the iteration mode of the target. Choices:

- `"vector"`: branching happens with `vctrs::vec_slice()` and aggregation happens with `vctrs::vec_c()`.
- `"list"`, branching happens with `[[]]` and aggregation happens with `list()`.
- `"group"`: `dplyr::group_by()`-like functionality to branch over subsets of a non-dynamic data frame. For `iteration = "group"`, the target must not by dynamic (the `pattern` argument of `tar_target()` must be left `NULL`). The target's return value must be a data frame with a special `tar_group` column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the `tar_group()` function to see how you can create the special `tar_group` column with `dplyr::group_by()`.

`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.

- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "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.

memory
: Character of length 1, memory strategy. If "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). If "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. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy 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, whether to run base::gc() just before the target runs.

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
: Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

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 of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

  - "main": the target's return value is sent back to the host machine and saved/uploaded locally.
  - "worker": the worker saves/uploads the value.
  - "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
    If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.
    The distinguishing feature of storage = "none" (as opposed to format = "file") is that in the general case, downstream targets will automatically

try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval` Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`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

See Also

`targets::tar_target_raw()`

Examples

```
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_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"),
  iteration = targets::tar_option_get("iteration"),
  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. 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 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.

pattern — Language to define branching for a target. For example, in a pipeline with numeric vector targets x and y, 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.

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.

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.

iteration — Character of length 1, name of the iteration mode of the target. Choices:

- "vector": branching happens with vctrs::vec_slice() and aggregation happens with vctrs::vec_c().
- "list", branching happens with [[]] and aggregation happens with list().
- "group": dplyr::group_by()-like functionality to branch over subsets of a non-dynamic data frame. For iteration = "group", the target must not by dynamic (the pattern argument of tar_target() must be left NULL). The target's return value must be a data frame with a special tar_group column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().

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.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "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.

memory
: Character of length 1, memory strategy. If "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). If "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. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy 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, whether to run base::gc() just before the target runs.

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
: Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

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 of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

  - "main": the target's return value is sent back to the host machine and saved/uploaded locally.
  - "worker": the worker saves/uploads the value.
  - "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").
    If you select storage = "none", then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (format = "file") it is the responsibility of the user to write to the data store from inside the target.

The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval` Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`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

Author(s)

Andrew Gene Brown

Nicholas Tierney

See Also

`targets::tar_target_raw()`

Examples

```
if (Sys.getenv("TAR_LONG_EXAMPLES") == "true") {
  targets::tar_dir({ # tar_dir() runs code from a temporary directory.
    library(geotargets)
    targets::tar_script({
      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()
    x <- targets::tar_read(raster_elevs)
  })
}
```

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

Description

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, potentially using parallel workers.

Usage

```
tar_terra_tiles(
  name,
  raster,
  ncol,
  nrow,
  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. 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 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.
raster	a `SpatRaster` object to be split into tiles
ncol	integer; number of columns to split the SpatRaster into
nrow	integer; number of rows to split the SpatRaster into
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. 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. • `"abridge"`: any currently running targets keep running, but no new targets launch after that. (Visit `https://books.ropensci.org/targets/debugging.html` to learn how to debug targets using saved workspaces.)

- `"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.

`memory` Character of length 1, memory strategy. If `"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). If `"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. For cloud-based dynamic files (e.g. `format = "file"` with `repository = "aws"`), this memory strategy 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, whether to run `base::gc()` just before the target runs.

`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` Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in `tar_make_future()`).

`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 of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's return value is sent back to the host machine and saved/uploaded locally.
- `"worker"`: the worker saves/uploads the value.
- `"none"`: almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when `retrieval = "none"`).
  If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.
  The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval` Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`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

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.

Author(s)

Eric Scott

Examples

```
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,
                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"),
  iteration = targets::tar_option_get("iteration"),
  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. 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 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.

`pattern` Language to define branching for a target. For example, in a pipeline with numeric vector targets `x` and `y`, `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.

`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 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.

`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.

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.

`iteration` Character of length 1, name of the iteration mode of the target. Choices:

- `"vector"`: branching happens with `vctrs::vec_slice()` and aggregation happens with `vctrs::vec_c()`.
- `"list"`, branching happens with `[[]]` and aggregation happens with `list()`.
- `"group"`: `dplyr::group_by()`-like functionality to branch over subsets of a non-dynamic data frame. For `iteration = "group"`, the target must not by dynamic (the `pattern` argument of `tar_target()` must be left `NULL`). The target's return value must be a data frame with a special `tar_group` column of consecutive integers from 1 through the number of groups. Each integer designates a group, and a branch is created for each collection of

rows in a group. See the tar_group() function to see how you can create the special tar_group column with dplyr::group_by().

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.
- "abridge": any currently running targets keep running, but no new targets launch after that. (Visit https://books.ropensci.org/targets/debugging.html to learn how to debug targets using saved workspaces.)
- "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.

memory — Character of length 1, memory strategy. If "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). If "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. For cloud-based dynamic files (e.g. format = "file" with repository = "aws"), this memory strategy 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, whether to run base::gc() just before the target runs.

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 — Numeric of length 1 between 0 and 1. Controls which targets get deployed first when multiple competing targets are ready simultaneously. Targets with priorities closer to 1 get dispatched earlier (and polled earlier in tar_make_future()).

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 of length 1, only relevant to tar_make_clustermq() and tar_make_future(). Must be one of the following values:

- "main": the target's return value is sent back to the host machine and saved/uploaded locally.
- "worker": the worker saves/uploads the value.
- "none": almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language. If you do use it, then the return value of the target is totally ignored when the target ends, but each downstream target still attempts to load the data file (except when retrieval = "none").

If you select `storage = "none"`, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (`format = "file"`) it is the responsibility of the user to write to the data store from inside the target.
The distinguishing feature of `storage = "none"` (as opposed to `format = "file"`) is that in the general case, downstream targets will automatically try to load the data from the data store as a dependency. As a corollary, `storage = "none"` is completely unnecessary if `format` is `"file"`.

`retrieval` Character of length 1, only relevant to `tar_make_clustermq()` and `tar_make_future()`. Must be one of the following values:

- `"main"`: the target's dependencies are loaded on the host machine and sent to the worker before the target runs.
- `"worker"`: the worker loads the targets dependencies.
- `"none"`: the dependencies are not loaded at all. This choice is almost never recommended. It is only for niche situations, e.g. the data needs to be loaded explicitly from another language.

`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

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 `GeoJSON` and `ESRI Shapefile` which both appear to work generally.

Examples

```
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)
})
}
```

Index