

Package: geotargets (via r-universe)

October 31, 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 7.3.2

Imports targets (>= 1.8.0), rlang (>= 1.1.3), cli (>= 3.6.2), terra (>= 1.7.71), withr (>= 3.0.0), zip

Suggests crew (>= 0.9.2), ncmeta, sf, stars, testthat (>= 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 d2fe65e28bce14f7bff487cf19b8dc7ca74e4b9e

Contents

geotargets_option_set	2
set_window	3
tar_stars	4

tar_terra_rast	9
tar_terra_sds	13
tar_terra_sprc	17
tar_terra_tiles	21
tar_terra_vect	25
tile_grid	29

Index	31
--------------	-----------

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,
    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/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>

`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.

`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"),
    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"),
  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 <code>targets::tar_target()</code> for more information.
command	R code to run the target.
pattern	Code to define a dynamic branching pattern for a target. See <code>targets::tar_target()</code> for more information.

proxy	logical. Passed to <code>stars::read_stars()</code> . If TRUE the target will be read as an object of class <code>stars_proxy</code> . Otherwise, the object is class <code>stars</code> .
mdim	logical. Use the Multidimensional Raster Data Model via <code>stars::write_mdim()</code> ? Default: FALSE. Only supported for some drivers, e.g. "netCDF" or "Zarr".
ncdf	logical. Use the NetCDF library directly to read data via <code>stars::read_ncdf()</code> ? Default: FALSE. Only supported for driver="netCDF".
driver	character. File format expressed as GDAL driver names passed to <code>stars::write_stars()</code> . See <code>sf::st_drivers()</code> .
options	character. GDAL driver specific datasource creation options passed to <code>stars::write_stars()</code> .
...	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 <code>!!</code> 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 <code>tar_option_set()</code> to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
repository	<p>Character of length 1, remote repository for target storage. Choices:</p> <ul style="list-style-type: none"> • "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 <code>tar_resources_aws()</code>, 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 <code>tar_repository_cas()</code> for content-addressable storage. <p>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.</p>
error	<p>Character of length 1, what to do if the target stops and throws an error. Options:</p> <ul style="list-style-type: none"> • "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. • "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. 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 <code>base::gc()</code> 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 <code>tar_make_future()</code>).
resources	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
storage	Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code> . Must be one of the following values: <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, 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.</p>

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	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • <code>"main"</code>: the target's dependencies are loaded on the host machine and sent to the worker before the target runs. • <code>"worker"</code>: the worker loads the targets dependencies. • <code>"none"</code>: 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 <code>tar_cue()</code> 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 <code>tar_manifest()</code> and <code>tar_visnetwork()</code> , and they let you select subsets of targets for the <code>names</code> argument of functions like <code>tar_make()</code> . For example, <code>tar_manifest(names = tar_described_as(starts_with("survival model")))</code> lists all the targets whose descriptions start with the character string <code>"survival model"</code> .

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

```
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"),
  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()
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.

...	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 <code>!!</code> 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 <code>tar_option_set()</code> 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: <ul style="list-style-type: none"> • "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 <code>tar_resources_aws()</code>, 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 <code>tar_repository_cas()</code> 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: <ul style="list-style-type: none"> • "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. • "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: <ol style="list-style-type: none"> 1. It is not downstream of the error, and 2. It is not a sibling branch from the same <code>tar_target()</code> 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. 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 <code>base::gc()</code> 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 <code>tar_make_future()</code>).
resources	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
storage	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, 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.</p> <p>The distinguishing feature of <code>storage = "none"</code> (as opposed to <code>format = "file"</code>) 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, <code>storage = "none"</code> is completely unnecessary if <code>format</code> is "file".</p>
retrieval	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.

	<ul style="list-style-type: none"> • "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 <code>tar_cue()</code> 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 <code>tar_manifest()</code> and <code>tar_visnetwork()</code> , and they let you select subsets of targets for the <code>names</code> argument of functions like <code>tar_make()</code> . For example, <code>tar_manifest(names = tar_described_as(starts_with("survival model")))</code> lists all the targets whose descriptions start with the character string "survival model".

Details

`terra::SpatRaster` objects do not contain raster 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 `tar_target()`. `tar_terra_rast()` handles this issue by writing and reading the target as a geospatial file (specified by `filetype`) rather than saving the `SpatRaster` object 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

```
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 <code>!!</code> 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 <code>tar_option_set()</code> to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
repository	<p>Character of length 1, remote repository for target storage. Choices:</p> <ul style="list-style-type: none"> • "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 <code>tar_resources_aws()</code>, 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 <code>tar_repository_cas()</code> for content-addressable storage. <p>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.</p>
error	<p>Character of length 1, what to do if the target stops and throws an error. Options:</p> <ul style="list-style-type: none"> • "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. • "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: <ol style="list-style-type: none"> 1. It is not downstream of the error, and 2. It is not a sibling branch from the same <code>tar_target()</code> call (if the error happened in a dynamic branch). <p>The idea is to avoid starting any new work that the immediate error impacts. <code>error = "trim"</code> is just like <code>error = "abridge"</code>, 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.)</p>
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.

<code>garbage_collection</code>	Logical, whether to run <code>base::gc()</code> just before the target runs.
<code>deployment</code>	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 .
<code>priority</code>	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 <code>tar_make_future()</code>).
<code>resources</code>	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
<code>storage</code>	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (<code>format = "file"</code>) it is the responsibility of the user to write to the data store from inside the target.</p> <p>The distinguishing feature of <code>storage = "none"</code> (as opposed to <code>format = "file"</code>) 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, <code>storage = "none"</code> is completely unnecessary if <code>format</code> is "file".</p>
<code>retrieval</code>	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>tar_cue()</code> 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 <code>tar_manifest()</code> and <code>tar_visnetwork()</code> , and they let you select subsets of targets for the <code>names</code> argument of functions like <code>tar_make()</code> . For example, <code>tar_manifest(names = tar_described_as(starts_with("survival model")))</code> 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.

Author(s)

Andrew Gene Brown

Nicholas Tierney

Eric R. Scott

See Also

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

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) {
        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 <code>!!</code> 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 <code>tar_option_set()</code> to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
repository	<p>Character of length 1, remote repository for target storage. Choices:</p> <ul style="list-style-type: none"> • "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 <code>tar_resources_aws()</code>, 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 <code>tar_repository_cas()</code> for content-addressable storage. <p>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.</p>
error	<p>Character of length 1, what to do if the target stops and throws an error. Options:</p> <ul style="list-style-type: none"> • "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. • "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: <ol style="list-style-type: none"> 1. It is not downstream of the error, and 2. It is not a sibling branch from the same <code>tar_target()</code> call (if the error happened in a dynamic branch). <p>The idea is to avoid starting any new work that the immediate error impacts. <code>error = "trim"</code> is just like <code>error = "abridge"</code>, 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.)</p>
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 <code>base::gc()</code> 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 <code>tar_make_future()</code>).
resources	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
storage	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, 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.</p> <p>The distinguishing feature of <code>storage = "none"</code> (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, <code>storage = "none"</code> is completely unnecessary if format is "file".</p>
retrieval	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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

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

```
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	<i>Split a raster into tiles that can be iterated over with dynamic branching</i>
-----------------	---

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 using 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	<p>a helper function that returns a list of numeric vectors such as <code>tile_grid</code> or <code>tile_blocksize</code> specified in one of the following ways:</p> <ul style="list-style-type: none"> • A named function, e.g. <code>tile_blocksize</code> or <code>"tile_blocksize"</code>. • An anonymous function, e.g. <code>\(x) tile_grid(x, nrow = 2, ncol = 2)</code>.
filetype	character. File format expressed as GDAL driver names passed to <code>terra::makeTiles()</code> .
gdal	character. GDAL driver specific datasource creation options passed to <code>terra::makeTiles()</code> .
...	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 <code>tar_option_set()</code> to set packages globally for all subsequent targets you define.
library	Character vector of library paths to try when loading packages.
repository	<p>Character of length 1, remote repository for target storage. Choices:</p> <ul style="list-style-type: none"> • <code>"local"</code>: file system of the local machine. • <code>"aws"</code>: Amazon Web Services (AWS) S3 bucket. Can be configured with a non-AWS S3 bucket using the endpoint argument of <code>tar_resources_aws()</code>, 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. • <code>"gcp"</code>: 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 <code>tar_repository_cas()</code> for content-addressable storage. <p>Note: if repository is not <code>"local"</code> and format is <code>"file"</code> 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.</p>
error	<p>Character of length 1, what to do if the target stops and throws an error. Options:</p> <ul style="list-style-type: none"> • <code>"stop"</code>: the whole pipeline stops and throws an error. • <code>"continue"</code>: the whole pipeline keeps going. • <code>"null"</code>: 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. • <code>"abridge"</code>: any currently running targets keep running, but no new targets launch after that. • <code>"trim"</code>: all currently running targets stay running. A queued target is allowed to start if: <ol style="list-style-type: none"> 1. It is not downstream of the error, and 2. It is not a sibling branch from the same <code>tar_target()</code> call (if the error happened in a dynamic branch). <p>The idea is to avoid starting any new work that the immediate error impacts. <code>error = "trim"</code> is just like <code>error = "abridge"</code>, 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.)</p>

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 <code>base::gc()</code> 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 <code>tar_make_future()</code>).
resources	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
storage	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (<code>format = "file"</code>) it is the responsibility of the user to write to the data store from inside the target.</p> <p>The distinguishing feature of <code>storage = "none"</code> (as opposed to <code>format = "file"</code>) 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, <code>storage = "none"</code> is completely unnecessary if <code>format</code> is "file".</p>
retrieval	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "main": the target's dependencies are loaded on the host machine and sent to the worker before the target runs.

	<ul style="list-style-type: none"> • "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 <code>tar_cue()</code> 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 <code>tar_manifest()</code> and <code>tar_visnetwork()</code> , and they let you select subsets of targets for the <code>names</code> argument of functions like <code>tar_make()</code> . For example, <code>tar_manifest(names = tar_described_as(starts_with("survival model")))</code> 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.

Note

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

Author(s)

Eric Scott

See Also

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

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"),
  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 <code>targets::tar_target()</code> for more information.
filetype	character. File format expressed as GDAL driver names passed to <code>terra::writeVector()</code> . See 'Note' for more details.
gdal	character. GDAL driver specific datasource creation options passed to <code>terra::writeVector()</code> .
...	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 <code>tar_option_set()</code> 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 <code>!!</code> to programmatically insert the values of global objects.
library	Character vector of library paths to try when loading packages.
repository	<p>Character of length 1, remote repository for target storage. Choices:</p> <ul style="list-style-type: none"> • "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 <code>endpoint</code> argument of <code>tar_resources_aws()</code>, 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 <code>tar_repository_cas()</code> for content-addressable storage. <p>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.</p>
error	<p>Character of length 1, what to do if the target stops and throws an error. Options:</p> <ul style="list-style-type: none"> • "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. • "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: <ol style="list-style-type: none"> 1. It is not downstream of the error, and 2. It is not a sibling branch from the same <code>tar_target()</code> 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. 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. <code>format = "file"</code> with <code>repository = "aws"</code>), 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 <code>base::gc()</code> 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 <code>tar_make_future()</code>).
resources	Object returned by <code>tar_resources()</code> with optional settings for high-performance computing functionality, alternative data storage formats, and other optional capabilities of targets. See <code>tar_resources()</code> for details.
storage	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>retrieval = "none"</code>). <p>If you select <code>storage = "none"</code>, then the return value of the target's command is ignored, and the data is not saved automatically. As with dynamic files (<code>format = "file"</code>) it is the responsibility of the user to write to the data store from inside the target.</p> <p>The distinguishing feature of <code>storage = "none"</code> (as opposed to <code>format = "file"</code>) 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, <code>storage = "none"</code> is completely unnecessary if <code>format</code> is "file".</p>

retrieval	<p>Character of length 1, only relevant to <code>tar_make_clustermq()</code> and <code>tar_make_future()</code>. Must be one of the following values:</p> <ul style="list-style-type: none"> • "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 <code>tar_cue()</code> 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 <code>tar_manifest()</code> and <code>tar_visnetwork()</code> , and they let you select subsets of targets for the names argument of functions like <code>tar_make()</code> . For example, <code>tar_manifest(names = tar_described_as(starts_with("survival model")))</code> lists all the targets whose descriptions start with the character string "survival model".

Details

`terra::SpatVector` objects do not contain vector 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 `tar_target()`. `tar_terra_rast()` handles this issue by writing and reading the target as a geospatial file (specified by `filetype`) rather than saving the `SpatVector` object 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 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)
})
}

```

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

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

Details

`tile_blocksize()` creates extents using the raster's native blocksize (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. `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).

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)

## Not run:
#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"
  )
)

## End(Not run)
```

Index

geotargets_option_get
 (geotargets_option_set), 2
geotargets_option_set, 2

set_window, 3

sf::st_drivers(), 6

stars::read_ncdf(), 6

stars::read_stars(), 6

stars::write_mdin(), 6

stars::write_stars(), 6

tar_make(), 8, 12, 16, 20, 24, 28

tar_make_clustermq(), 7, 8, 11, 15, 19, 23, 27, 28

tar_make_future(), 7, 8, 11, 15, 19, 23, 27, 28

tar_manifest(), 8, 12, 16, 20, 24, 28

tar_repository_cas(), 6, 10, 14, 18, 22, 26

tar_resources_aws(), 6, 10, 14, 18, 22, 26

tar_stars, 4

tar_stars_proxy (tar_stars), 4

tar_target(), 7, 10, 14, 18, 22, 26

tar_terra_rast, 9

tar_terra_rast(), 24

tar_terra_sds, 13

tar_terra_sprc, 17

tar_terra_sprc(), 16

tar_terra_tiles, 21

tar_terra_tiles(), 4, 29

tar_terra_vect, 25

tar_visnetwork(), 8, 12, 16, 20, 24, 28

targets::tar_target(), 5, 8, 9, 12, 13, 17, 21, 25, 26

targets::tar_target_raw(), 16, 20

terra::ext(), 30

terra::fileBlocksize(), 29

terra::getTileExtents(), 29

terra::makeTiles(), 22

terra::SpatRaster, 9, 12

terra::SpatRasterCollection, 17

terra::SpatRasterDataset, 13

terra::SpatVector, 25, 28

terra::window(), 3

terra::writeRaster(), 3, 9, 13, 17

terra::writeVector(), 26

tile_blocksize, 22

tile_blocksize(tile_grid), 29

tile_blocksize(), 24

tile_grid, 22, 29

tile_grid(), 24

tile_n(tile_grid), 29

tile_n(), 24