Package: xyt (via r-universe)

November 5, 2026

Type Package

Title Extract Raster Time Series Values at Points in Space and Time

Version 0.1.0

Description Extract values from a time series of rasters at points that each carry their own date-time. The series is described by a reader function rather than by data held in memory, so only the time slices actually needed are read. Values are taken from the slice nearest in time, or interpolated linearly between the two slices that bracket each point. Derived from the extract() method of package 'raadtools'.

Depends R (>= 4.1.0)

Imports terra (>= 1.7.0)

Suggests knitr, mirai (>= 1.3.0), mori (>= 0.2.0), rmarkdown, testthat (>= 3.0.0)

License GPL-3

Encoding UTF-8

VignetteBuilder knitr

Roxygen list(markdown = TRUE)

Config/testthat/edition 3

Config/roxygen2/version 8.1.0

URL https://github.com/AustralianAntarcticDivision/xyt, https://australianantarcticdivision.github.io/xyt/

BugReports https://github.com/AustralianAntarcticDivision/xyt/issues

Config/pak/sysreqs libgdal-dev gdal-bin libgeos-dev libproj-dev libsqlite3-dev

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

Date/Publication 2026-09-24 06:16:40 UTC

RemoteUrl https://github.com/AustralianAntarcticDivision/xyt

RemoteRef HEAD

RemoteSha a7766adadaac19ba4ba6245860113c165aef17c4

Contents

extract_xyt 2
synthetic_reader 4
synthetic_value 5
xyt_map_mirai 6
xyt_reader_check 7
xyt_source 7

Index **9**

`extract_xyt` *Extract values from a raster time series at points in space and time*

Description

Each row of `xyt` is a location and a date-time. The value returned for it is read from the time slice nearest that date-time, or, with `ctstime = TRUE`, interpolated linearly between the two slices that bracket it.

Usage

```
extract_xyt(
  read,
  xyt,
  ctstime = FALSE,
  when = c("nearest", "previous"),
  method = c("simple", "bilinear"),
  fact = NULL,
  crs = "EPSG:4326",
  files = NULL,
  share = NULL,
  tolerance = NULL,
  map = NULL,
  verbose = interactive(),
  ...
)
```

Arguments

`read`	a reader function. See the reader contract in `xyt`, `xyt_reader_check()` and `xyt_source()`.
`xyt`	a data.frame or matrix with three columns: x, y and time. Extra columns are ignored. The time column may be POSIXct, Date or character.
`ctstime`	interpolate linearly in time between bracketing slices (`TRUE`), or take the value from one slice (`FALSE`, the default).
`when`	which slice a point takes its value from when `ctstime = FALSE`: the closest in time (`"nearest"`, the default) or the most recent one at or before it (`"previous"`).

`method`	how the value is taken from the raster: `"simple"` for the value of the cell the point falls in, `"bilinear"` for a distance weighted average of the four nearest cell centres.
`fact`	optional aggregation factor applied to each slice before extraction, as in `terra::aggregate()`. One integer, or two for the horizontal and vertical factors.
`crs`	coordinate reference system of the coordinates in `xyt`, defaulting to longitude/latitude on WGS84. Points are transformed to the raster's own system before extraction.
`files`	the catalogue, if you already have it: a data.frame with a `date` column. Supplied here, `read(returnfiles = TRUE)` is never called, so a reader only has to do the one job of reading a slice. See also `xyt_source()`.
`share`	put the catalogue into shared memory with `mori::share()`, so that it is mapped by the daemons rather than copied to each of them. `NULL`, the default, does it when mori is installed and mirai daemons are running, and otherwise leaves the catalogue alone. `TRUE` insists, `FALSE` refuses.
`tolerance`	how far, in days, a point may sit from the slice matched to it before its value is refused and returned as `NA`. `NULL`, the default, derives it from the spacing of the series.
`map`	how the per-slice reads are run. `NULL`, the default, uses mirai daemons if any are running and reads serially otherwise. Pass `base::lapply()` to force serial, or see `xyt_map_mirai()`.
`verbose`	report progress as slices are read.
`...`	passed to `read`, and only to `read`.

Details

Only the slices that some point actually needs are read, and each of those is read exactly once.

`...` reaches the reader alone. This is a deliberate departure from the version of this code that lived in raadtools, where `...` was passed to both the reader and the extraction, so that an argument meant for one was silently offered to the other. Arguments controlling the extraction are named here instead.

Value

a numeric vector with one value per row of `xyt`.

Examples

```
read <- synthetic_reader()
xyt <- data.frame(x = c(0.5, 2.5), y = c(0.5, 3.5),
                  t = as.POSIXct(c("2000-01-01", "2000-01-05"), tz = "UTC"))
extract_xyt(read, xyt)
extract_xyt(read, xyt, ctstime = TRUE)
```

`synthetic_reader` *A reader function with no data behind it*

Description

Builds a reader that satisfies the contract `extract_xyt()` expects, over a small in-memory series whose values are known in closed form. It exists so the extraction machinery can be exercised, and its answers checked against arithmetic, without any files, network or credentials.

Usage

```
synthetic_reader(
  n = 10L,
  start = "2000-01-01",
  by = "1 day",
  crs = "EPSG:4326",
  extent = c(0, 4, 0, 4),
  res = 1
)
```

Arguments

`n`	number of time slices.
`start`	date-time of the first slice.
`by`	spacing between slices, as understood by `seq.POSIXt()`.
`crs`	coordinate reference system of the slices.
`extent`	extent of the slices, as xmin, xmax, ymin, ymax.
`res`	cell size.

Details

Each slice holds the value

```
value = 100 * x + y + i
```

where `x` and `y` are the coordinates of the cell centre and `i` is the one-based position of the slice in the series. Two properties follow, and both are used in the tests. The field is linear in `x` and `y`, so `method = "bilinear"` returns `100 * x + y + i` exactly anywhere in the interior. And it is linear in `i`, so `ctstime = TRUE` at a fraction `p` between slices `i` and `i + 1` returns exactly `100 * x + y + i + p`.

Value

a function of the form `read(date, returnfiles = FALSE, inputfiles = NULL, offset = 0, ...)`. `offset` is added to every value, and is there so a test can prove that `...` reaches the reader.

Examples

```
read <- synthetic_reader()
head(read(returnfiles = TRUE))
read(as.POSIXct("2000-01-03", tz = "UTC"))
```

`synthetic_value` *Value of the synthetic series, computed directly*

Description

The closed form of what `synthetic_reader()` holds, for writing expected values in tests without going through a raster.

Usage

```
synthetic_value(
  x,
  y,
  i,
  method = c("simple", "bilinear"),
  res = 1,
  origin = c(0, 0)
)
```

Arguments

`x, y`	coordinates. For `method = "simple"` these are snapped to the containing cell centre; for `"bilinear"` they are used as given.
`i`	slice number, one-based. May be fractional, which is what `ctstime = TRUE` returns.
`method`	`"simple"` or `"bilinear"`.
`res`	cell size of the series.
`origin`	lower left corner of the series.

Value

a numeric vector.

Examples

```
synthetic_value(0.7, 2.2, i = 3)
synthetic_value(0.7, 2.2, i = 3, method = "bilinear")
```

`xyt_map_mirai` *Run the per-slice reads on mirai daemons*

Description

The work `extract_xyt()` does divides cleanly: one slice is read, the points that want it are extracted, and a plain numeric vector comes back. Nothing is shared between slices and no SpatRaster crosses a process boundary, so the reads can be spread over daemons with no change to the answer.

Usage

```
xyt_map_mirai(...)
```

Arguments

`...`	passed to `mirai::mirai_map()`.

Details

Start daemons in the usual way and `extract_xyt()` will use them without being told to:

```
mirai::daemons(6)
mirai::everywhere({ library(raadtools) })   # whatever the reader needs
extract_xyt(read, xyt)
mirai::daemons(0)
```

`mirai::everywhere()` matters: a daemon runs the reader in a fresh session, so any package the reader reaches for has to be loaded there. The reader itself is sent along with the task.

Whether this is faster depends on where the bytes come from. Reading a hundred slices off a remote store is latency bound and parallelises well. Reading ten slices off a local disk that is already saturated will not.

Value

a function of `(X, FUN)`, for `extract_xyt()`'s `map` argument.

See Also

`extract_xyt()`

Examples

```
if (requireNamespace("mirai", quietly = TRUE)) {
  ## a mapper to hand to extract_xyt(map = )
  mapper <- xyt_map_mirai()
  mapper
}
```

xyt_reader_check *Check a function against the reader contract*

Description

Runs the calls `extract_xyt()` will make, in the order it will make them, and reports what came back. Use it when wiring up a new source, so a mismatch is a sentence about the contract rather than an error from inside the extraction loop.

Usage

```
xyt_reader_check(read, error = TRUE, ...)
```

Arguments

`read`	the function to check.
`error`	stop on failure (the default), or return the report quietly.
`...`	passed to `read`, exactly as `extract_xyt()` would pass it.

Value

invisibly, a data.frame with one row per check: `check`, `ok` and `note`.

Examples

```
xyt_reader_check(synthetic_reader())
```

xyt_source *Build a reader from a catalogue and a slice function*

Description

The reader contract is one function that does two jobs, switched by `returnfiles`. That shape exists so that functions written before this package, raadtools' `read*` among them, satisfy it without being touched. It is not a nice thing to write by hand: the return type depends on the value of an argument, which is awkward to document and easy to get subtly wrong.

Usage

```
xyt_source(slice, catalogue)
```

Arguments

`slice`	a function of `(date, files, ...)` returning a single-layer SpatRaster for `date`. `files` is the catalogue.
`catalogue`	the catalogue: a data.frame with a `date` column, or a function returning one. A function is called once, the first time it is needed, and the result is reused for the life of the source. Build a separate source if you need a different catalogue.

Details

`xyt_source()` writes it for you. You supply the two jobs separately, each as an ordinary function with an ordinary signature, and get back something `extract_xyt()` accepts.

Value

a function of class `xyt_reader` satisfying the reader contract.

See Also

`extract_xyt()` takes a `files` argument too, which is the other way to avoid writing the flag: hand it the catalogue directly and the `returnfiles` call is never made.

Examples

```
## a series of two rasters held in a list
r <- lapply(1:2, function(i) {
  x <- terra::rast(terra::ext(0, 4, 0, 4), resolution = 1, crs = "EPSG:4326")
  terra::values(x) <- i
  x
})
src <- xyt_source(
  slice = function(date, files, ...) r[[which(files$date == date)]],
  catalogue = data.frame(date = as.POSIXct(c("2000-01-01", "2000-01-02"),
                                           tz = "UTC"))
)
extract_xyt(src, data.frame(x = 1, y = 1,
                            t = as.POSIXct("2000-01-02", tz = "UTC")))
```

Index

base::lapply(), *3*

extract_xyt, 2
extract_xyt(), *4*, *6–8*

mirai::mirai_map(), *6*
mori::share(), *3*

seq.POSIXt(), *4*
synthetic_reader, 4
synthetic_reader(), *5*
synthetic_value, 5

terra::aggregate(), *3*

xyt, *2*
xyt_map_mirai, 6
xyt_map_mirai(), *3*
xyt_reader_check, 7
xyt_reader_check(), *2*
xyt_source, 7
xyt_source(), *2, 3*