

Package: xbioclim (via r-universe)

October 7, 2026

Type Package

Title Bioclimatic Variables from Monthly Climate Data

Version 1.0.3

Description Computes the 19 standard bioclimatic variables (BIO01-BIO19) from monthly climate data. The variable set was originally proposed by Nix (1986, ISBN:978-0-644-04887-3) for the BIOCLIM modelling system and is also distributed with the CHELSA climatologies (Karger et al., 2017 [doi:10.1038/sdata.2017.122](https://doi.org/10.1038/sdata.2017.122)). Provides both individual variable functions and a unified interface to compute all 19 variables at once. Designed as an R implementation of the 'xbioclim' C++ library (Robles Fernandez, 2026 <https://github.com/alroble/xbioclimcpp>). Supports single-pixel vectors and block-based raster processing via 'terra' for memory-efficient handling of large spatial datasets. Includes helpers to transform ERA5-Land hourly reanalysis data (Muñoz-Sabater et al., 2021 [doi:10.5194/essd-13-4349-2021](https://doi.org/10.5194/essd-13-4349-2021)) into monthly climate inputs.

License MIT + file LICENSE

NeedsCompilation yes

SystemRequirements GNU make, C++17; optionally GDAL (>= 2.0.1) with gdal-config, CUDA toolkit (>= 11.0) with nvcc for GPU acceleration

Encoding UTF-8

Language en-US

Roxygen list(markdown = TRUE)

Imports methods, Rcpp (>= 1.0.0)

LinkingTo Rcpp

Suggests parallel, sf, terra, testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

URL <https://alroble.github.io/xbioclim/>,
<https://github.com/alroble/xbioclim>

BugReports <https://github.com/alrobles/xbioclim/issues>

Collate 'ReppExports.R' 'primitives.R' 'bioclim.R' 'BioclimData.R'
 'quarterly.R' 'bioclim_window.R' 'bioclim_module.R'
 'bioclimmodel.R' 'bioclim_raster.R' 'era5_monthly.R'
 'era5_bioclim.R' 'bioclim_engine.R' 'engine.R' 'mask.R'
 'messages.R' 'xbioclim-package.R'

Config/roxygen2/version 8.0.0

Config/pak/sysreqs nvidia-cuda-dev libgdal-dev gdal-bin make

Repository <https://alrobles.r-universe.dev>

Date/Publication 2026-09-23 19:33:39 UTC

RemoteUrl <https://github.com/alrobles/xbioclim>

RemoteRef HEAD

RemoteSha 95f4301963df94bd4cf3a74cc77b1b60d281b1fd

Contents

xbioclim-package	3
apply_mask_cpp	4
bioclim-raster	5
bioclim-variables	5
bioclim_block	8
bioclim_engine	8
bioclim_errors	11
bioclim_raster	12
bioclim_rolling	13
bioclim_warnings	14
bioclim_window	15
BioclimData	16
BioclimData-class	17
BioclimModel	17
BioclimModel-class	18
BioclimModel-methods	19
clear_messages	19
ClimateBlock	20
create_mask	21
cuda_device_count	22
cuda_device_info	23
cuda_info	23
engine_compute	24
engine_create	24
engine_open	25
engine_set_device	26
engine_set_mask	26
engine_set_output	27
engine_set_threads	27

engine_set_variables	28
era5_bioclim	28
era5_bioclim_years	30
gdal_can_open	31
gdal_info	32
has_cuda	33
has_error	33
has_gdal	34
has_warning	34
quarterly_fixed	35
quarterly_rolling	36
quarterly_variables	36
rasterize_mask_cpp	37
xbioclim-messages	39

Index	40
--------------	-----------

xbioclim-package	<i>xbioclim: Bioclimatic Variables from Monthly Climate Data</i>
------------------	--

Description

Computes the 19 standard bioclimatic variables (BIO01-BIO19) from monthly climate data following the WorldClim specification. This is an R implementation of the xbioclim C++ library, with a compiled C++ back-end exposed through Rcpp Modules.

Value

No return value; package-level documentation.

ERA5-Land monthly aggregation

In addition to the BIO01–BIO19 computation, xbioclim provides helpers to aggregate ERA5-Land hourly t2m/tp into the CHELSA-compatible monthly variables (tas, tasmax, tasmin, pr) used by the bioclim functions:

- `era5_to_monthly()` – single-pass hourly → monthly aggregation.
- `era5_t2m_to_monthly()` / `era5_tp_to_monthly()` – variable-specific helpers.
- `era5_bioclim()` / `era5_bioclim_years()` – end-to-end ERA5-Land → BIO pipeline.

Error and warning propagation

xbioclim mirrors the SpatMessages pattern used by the terra package. C++ routines record errors and warnings into an internal message store rather than throwing directly. R-side wrappers around C++ calls should invoke `check_messages()` after each call to convert any stored messages into native R conditions. Users can also inspect the store programmatically:

- `bioclim_errors()` / `bioclim_warnings()` – retrieve stored messages.
- `has_error()` / `has_warning()` – test whether messages exist.
- `clear_messages()` – reset the store.

Author(s)

Maintainer: Angel Luis Robles Fernandez <a.l.robles.fernandez@gmail.com>

Authors:

- Angel Luis Robles Fernandez <a.l.robles.fernandez@gmail.com>

See Also

Useful links:

- <https://alrobles.github.io/xbioclim/>
- <https://github.com/alrobles/xbioclim>
- Report bugs at <https://github.com/alrobles/xbioclim/issues>

apply_mask_cpp

Apply a binary mask raster to an input raster

Description

Reads input_path and mask_path tile-by-tile (one row at a time) and writes the result to output_path. Pixels where the mask equals 0 are replaced with NaN in the output; all other pixels retain their original values (with any GDAL scale/offset applied).

Usage

```
apply_mask_cpp(input_path, mask_path, output_path)
```

Arguments

input_path	Character string: path to a GDAL-readable raster (any number of bands).
mask_path	Character string: path to a single-band GDT_Byte mask raster (e.g., produced by rasterize_mask_cpp).
output_path	Character string: file path where the output Float64 GeoTIFF will be written (created or overwritten).

Details

This function is the low-level C++ entry point. Most users should call the higher-level [create_mask](#) wrapper instead.

Stops with an informative error if the package was built without GDAL support or if the mask and input dimensions differ.

Value

Invisibly returns NULL. The side effect is the creation of the masked raster at output_path.

See Also

[create_mask](#), [rasterize_mask_cpp](#)

Examples

```
if (has_gdal()) {
  # Requires GDAL support at build time.
  ref  <- system.file("extdata", "tiny.tif", package = "xbioclim")
  poly <- tempfile(fileext = ".geojson")
  mask <- tempfile(fileext = ".tif")
  output <- tempfile(fileext = ".tif")
  writeLines(
    '{"type": "FeatureCollection", "features": [{"type": "Feature",
      "geometry": {"type": "Polygon", "coordinates": [[[0,0],[1,0],[1,1],[0,1],[0,0]]]},
      "properties": {}}]}' ,
    poly)
  rasterize_mask_cpp(poly, ref, mask)
  apply_mask_cpp(ref, mask, output)
}
```

bioclim-raster	<i>Block-Based Raster Processing for Bioclimatic Variables</i>
----------------	--

Description

Functions for computing bioclimatic variables from monthly climate rasters (SpatRaster objects) using terra's block-loop architecture for memory-efficient processing of large rasters.

Value

No return value; this is an overview page. [bioclim_raster\(\)](#) returns a [SpatRaster](#) with 19 layers named bio01 through bio19.

bioclim-variables	<i>Compute Bioclimatic Variables from Monthly Climate Data</i>
-------------------	--

Description

Computes the 19 standard bioclimatic variables (BIO01-BIO19) from monthly climate data following the WorldClim specification. This is an R implementation of the xbioclim C++ library.

Usage

```
bio01(tas, ...)  
  
bio02(tasmax, tasmin, ...)  
  
bio03(tasmax, tasmin, ...)  
  
bio04(tas, ...)  
  
bio05(tasmax, ...)  
  
bio06(tasmin, ...)  
  
bio07(tasmax, tasmin, ...)  
  
bio08(tas, pr, ...)  
  
bio09(tas, pr, ...)  
  
bio10(tas, ...)  
  
bio11(tas, ...)  
  
bio12(pr, ...)  
  
bio13(pr, ...)  
  
bio14(pr, ...)  
  
bio15(pr, ...)  
  
bio16(pr, ...)  
  
bio17(pr, ...)  
  
bio18(tas, pr, ...)  
  
bio19(tas, pr, ...)  
  
bioclim(tas, tasmax, tasmin, pr, ...)
```

Arguments

tas	Numeric vector of length 12 or BioclimData : monthly mean temperature.
...	Additional arguments. For BioclimData inputs the argument <code>na.rm</code> is accepted: if TRUE, missing months are omitted and each BIO is computed from the available months (a quarter needs at least one valid month). The default <code>na.rm = FALSE</code> makes a pixel all-NA if any input month is NA.

tasmax	Numeric vector of length 12: monthly maximum temperature.
tasmin	Numeric vector of length 12: monthly minimum temperature.
pr	Numeric vector of length 12: monthly precipitation.

Details

The 19 bioclimatic variables are:

- BIO01: Mean Annual Temperature
- BIO02: Mean Diurnal Range (mean of monthly (tasmax - tasmin))
- BIO03: Isothermality ($100 * \text{BIO02} / \text{BIO07}$)
- BIO04: Temperature Seasonality ($100 * \text{population SD of monthly tas}$)
- BIO05: Max Temperature of Warmest Month
- BIO06: Min Temperature of Coldest Month
- BIO07: Temperature Annual Range (BIO05 - BIO06)
- BIO08: Mean Temperature of Wettest Quarter
- BIO09: Mean Temperature of Driest Quarter
- BIO10: Mean Temperature of Warmest Quarter
- BIO11: Mean Temperature of Coldest Quarter
- BIO12: Annual Precipitation
- BIO13: Precipitation of Wettest Month
- BIO14: Precipitation of Driest Month
- BIO15: Precipitation Seasonality (CV)
- BIO16: Precipitation of Wettest Quarter
- BIO17: Precipitation of Driest Quarter
- BIO18: Precipitation of Warmest Quarter
- BIO19: Precipitation of Coldest Quarter

All functions accept either plain numeric vectors of length 12 (single pixel) or a [BioclimData](#) object holding a raster block (multiple pixels). When a [BioclimData](#) object is supplied, computation is delegated to the compiled C++ backend (xbioclim) and a numeric vector (one value per pixel) is returned.

Value

- For bio01–bio19 with plain numeric-vector inputs: a single numeric value.
- For bioclim() with plain numeric-vector inputs: a named numeric vector of length 19 (names bio01–bio19).
- For [BioclimData](#) inputs to bio01–bio19: a numeric vector with one value per pixel.
- For [BioclimData](#) inputs to bioclim(): a numeric matrix with one row per pixel and 19 named columns (bio01–bio19).

bioclim_block	Create a C++ ClimateBlock and compute bioclimatic variables
---------------	---

Description

A convenience wrapper around the C++ ClimateBlock class exposed by the bioclim_mod Rcpp module. Accepts the same four monthly climate vectors as `bioclim` but delegates the computation to the compiled C++ back-end via Rcpp.

Usage

```
bioclim_block(tas, tasmax, tasmin, pr)
```

Arguments

tas	Numeric vector of length 12: monthly mean temperature.
tasmax	Numeric vector of length 12: monthly maximum temperature.
tasmin	Numeric vector of length 12: monthly minimum temperature.
pr	Numeric vector of length 12: monthly precipitation.

Value

A named numeric vector of length 19 (bio01 through bio19), identical in meaning to the output of `bioclim`.

Examples

```
tas    <- c(5, 7, 10, 14, 18, 22, 25, 24, 20, 15, 10, 6)
tasmax <- c(8, 10, 14, 18, 23, 28, 32, 31, 26, 19, 13, 9)
tasmin <- c(1, 3, 6, 10, 13, 17, 20, 19, 15, 10, 6, 2)
pr     <- c(60, 55, 50, 40, 30, 15, 5, 10, 25, 45, 55, 65)
bioclim_block(tas, tasmax, tasmin, pr)
```

bioclim_engine	Compute Bioclimatic Variables via the Native GDAL-Tiled Engine
----------------	--

Description

High-level R interface to the BioclimEngine C++ tiled computation pipeline. Reads four sets of monthly climate rasters (mean temperature, maximum temperature, minimum temperature, and precipitation) from disk, computes the requested bioclimatic variables, and writes all 19 variables to a single multi-band GeoTIFF named `bio.tif` inside the output directory — one tile at a time so that peak memory is proportional to `tile_size`, not the full raster extent.

Usage

```

bioclim_engine(
  tas,
  tasmax,
  tasmin,
  pr,
  output = tempfile("bioclim_"),
  variables = 1:19,
  mask = NULL,
  threads = 1L,
  tile_size = 256L,
  overwrite = FALSE,
  device = c("auto", "cpu", "gpu"),
  dtype = c("Float64", "Float32"),
  use_pipeline = FALSE
)

```

Arguments

tas	Character vector of length 1 (12-band file) or 12 (one file per month), or a terra::SpatRaster with 12 layers: monthly mean temperature.
tasmax	Like tas but for monthly maximum temperature.
tasmin	Like tas but for monthly minimum temperature.
pr	Like tas but for monthly precipitation.
output	Character string: path to the output directory where the multi-band GeoTIFF bio.tif will be written. The directory is created automatically if it does not exist. Defaults to a temporary directory.
variables	Integer vector of variable numbers to compute, with values in 1:19. Default is 1:19 (all 19 variables). For example, c(1, 12) returns only BIO01 and BIO12 from the 19-band output.
mask	Optional mask: a character file path, an sf object (polygon), or a terra::SpatRaster. NULL (default) means no masking.
threads	Positive integer: number of OpenMP threads to use. Default is 1L.
tile_size	Positive integer: tile dimension (pixels) for tiled I/O. Default is 256L.
overwrite	Logical: whether to overwrite an existing bio.tif inside output. Default is FALSE.
device	Character scalar: compute device to use. One of "auto" (default), "cpu", or "gpu". "auto" selects the GPU when a CUDA device is available, otherwise falls back to the CPU. "gpu" on a system without CUDA emits a warning and falls back to the CPU. When device = "gpu" and tile_size is left at its default, the tile size is automatically scaled to match the detected GPU memory (4096 for high-memory GPUs such as the A100, 1024 otherwise).
dtype	Character scalar: output data type, one of "Float64" (default) or "Float32".
use_pipeline	Logical scalar: if TRUE, use the experimental overlapped read/compute/write pipeline with parallel 12-reader I/O. Default is FALSE, which keeps the original serial tiled loop.

Details

GDAL requirement. This function requires the package to have been compiled with GDAL support (see [has_gdal](#)). If GDAL is not available an informative error is raised immediately.

Variable selection. By default all 19 standard bioclimatic variables (BIO01–BIO19) are returned. Pass variables as an integer vector (e.g. `c(1, 12, 15)`) to restrict the returned `terra::SpatRaster` to a subset. The engine always computes all 19 internally and writes a full 19-band `bio.tif`; the subsetting is applied when constructing the returned object.

Single multi-band output. The output directory contains one file, `bio.tif`, with 19 bands (BIO01–BIO19). This reduces GDAL I/O call overhead compared with the previous one-file-per-variable layout.

Tiled processing. The engine reads and writes rasters in square tiles of `tile_size × tile_size` pixels. Choosing a large tile improves I/O efficiency; a small tile reduces peak RAM. The default (256) is a good balance for most use cases.

Multi-band vs. single-band inputs. Each climate variable can be supplied either as twelve single-band files (one per calendar month) or as one multi-band file with exactly 12 bands. A `terra::SpatRaster` with 12 layers is also accepted; its on-disk source paths are extracted automatically via `terra::sources()`.

Mask support. An optional raster or vector mask can be used to restrict computation to a specific region. Pixels outside the mask are written as NaN. Accepted formats:

- character — path to any GDAL-readable raster or OGR vector.
- sf object — written to a temporary GeoJSON and rasterized.
- `terra::SpatRaster` — written to a temporary GeoTIFF.

Output data type. Use `dtype = "Float32"` to halve the output file size. Values are rounded from double-precision internal arithmetic to single precision on write; numerical differences are typically below $1e-5$.

Output. If **terra** is installed the function returns a `terra::SpatRaster` whose layers correspond to the selected variables. Otherwise it returns the path to the output `bio.tif` file.

Value

If **terra** is installed, a `terra::SpatRaster` with one layer per selected variable (named `bio01 ... bio19`). Otherwise a character scalar: the path to `bio.tif`.

See Also

[bioclim_raster](#) for the in-memory R/terra path, [has_gdal](#) to check GDAL availability, [engine_create](#) for the low-level XPtr interface.

Examples

```
if (has_gdal() && requireNamespace("terra", quietly = TRUE)) {
  library(terra)

  # Create tiny synthetic climate rasters (10x10 pixels, 12 layers each)
  make_rast <- function(vals, file) {
    r <- rast(nrows = 10, ncols = 10, nlyrs = 12,
```

```

      xmin = 0, xmax = 1, ymin = 0, ymax = 1, crs = "EPSG:4326")
    for (m in seq_len(12)) values(r[[m]]) <- vals[m]
    writeRaster(r, file, overwrite = TRUE)
    file
  }

tmp <- tempdir()
tas_file <- make_rast(c(5,7,10,14,18,22,25,24,20,15,10,6),
  file.path(tmp, "tas.tif"))
tasmax_file <- make_rast(c(8,10,14,18,23,28,32,31,26,19,13,9),
  file.path(tmp, "tasmax.tif"))
tasmin_file <- make_rast(c(1,3,6,10,13,17,20,19,15,10,6,2),
  file.path(tmp, "tasmin.tif"))
pr_file <- make_rast(c(60,55,48,35,28,22,18,20,35,55,65,68),
  file.path(tmp, "pr.tif"))

# Compute all 19 variables (single multi-band output file)
out_dir <- file.path(tmp, "bioclim_out")
result <- bioclim_engine(tas_file, tasmax_file, tasmin_file, pr_file,
  output = out_dir, overwrite = TRUE)

nlyr(result) # 19
list.files(out_dir, pattern = "[.]tif$")

# Compute only BIO01 and BIO12
out_dir2 <- file.path(tmp, "bioclim_subset")
result2 <- bioclim_engine(tas_file, tasmax_file, tasmin_file, pr_file,
  output = out_dir2, variables = c(1L, 12L),
  overwrite = TRUE)

nlyr(result2) # 2
names(result2) # "bio01" "bio12"
}

```

bioclim_errors	<i>Retrieve stored error messages</i>
----------------	---------------------------------------

Description

Returns the character vector of error messages currently held in the xbioclim message store. Under normal usage the store is automatically flushed by [check_messages\(\)](#) after every C++ call, but you can inspect it manually before that point if needed.

Usage

```
bioclim_errors()
```

Value

A character vector (possibly empty).

See Also

[bioclim_warnings\(\)](#), [has_error\(\)](#), [clear_messages\(\)](#)

Examples

```
clear_messages()
bioclim_errors() # character(0)
```

bioclim_raster

Compute Bioclimatic Variables from Monthly Climate Rasters

Description

Computes the 19 standard bioclimatic variables (BIO01-BIO19) from monthly climate raster data (SpatRaster objects with 12 layers, one per month) using terra's block-loop architecture for memory-efficient processing.

Usage

```
bioclim_raster(
  tas,
  tasmax,
  tasmin,
  pr,
  filename = "",
  n_blocks = NULL,
  ncores = 1L,
  overwrite = FALSE,
  ...
)
```

Arguments

tas	SpatRaster with 12 layers: monthly mean temperature.
tasmax	SpatRaster with 12 layers: monthly maximum temperature.
tasmin	SpatRaster with 12 layers: monthly minimum temperature.
pr	SpatRaster with 12 layers: monthly precipitation.
filename	Character string: output file path. Pass "" (default) to keep the result in memory.
n_blocks	Integer: target number of row blocks. If NULL (default), terra selects an appropriate number based on available memory.
ncores	Integer: number of CPU cores for within-block parallel processing via OpenMP. Default is 1 (sequential).
overwrite	Logical: whether to overwrite an existing output file. Default is FALSE.
...	Additional arguments passed to terra::writeStart().

Details

This function follows terra's block-loop pattern: the input rasters are read one horizontal block at a time (using `terra::readStart()`, `terra::readValues()`, and `terra::readStop()`), keeping peak memory use proportional to the block size rather than the full raster extent. Results are written to the output raster block by block (using `terra::writeStart()`, `terra::writeValues()`, and `terra::writeStop()`).

When `ncores > 1`, pixels within each block are processed in parallel using OpenMP threads via `bioclim_cpp()`. Only one block is held in memory at a time regardless of the number of threads.

Value

A [SpatRaster](#) with 19 layers named `bio01` through `bio19`, sharing the spatial extent, resolution and CRS of `tas`.

See Also

[bioclim\(\)](#) for single-pixel (vector) computation.

Examples

```
library(terra)

# Create small test rasters: 4 rows x 3 cols, 12 monthly layers
make_rast <- function(vals) {
  r <- rast(nrows = 4, ncols = 3, nlyr = 12)
  values(r) <- vals
  r
}

n <- 4 * 3 # number of cells
tas <- make_rast(matrix(rep(1:12, each = n), nrow = n, ncol = 12))
tasmax <- make_rast(matrix(rep(2:13, each = n), nrow = n, ncol = 12))
tasmin <- make_rast(matrix(rep(0:11, each = n), nrow = n, ncol = 12))
pr <- make_rast(matrix(rep(1:12, each = n), nrow = n, ncol = 12))

# Compute all 19 bioclimatic variables in a single pass
result <- bioclim_raster(tas, tasmax, tasmin, pr)
nlyr(result) # 19
```

Description

Computes the 19 bioclimatic variables using a rolling window of arbitrary length over the full 12 months. The base variables (`bio01..bio07`, `bio12..bio15`) are annual, while the rolling variables (`bio08..bio11`, `bio16..bio19`) use the best window-month period. With `window = 3` this is equivalent to the standard BIO08-BIO19.

Usage

```
bioclim_rolling(tas, tasmax, tasmin, pr, window = 3L, na.rm = FALSE, ...)
```

Arguments

tas	Numeric vector (length 12), matrix (pixels x 12), SpatRaster with 12 layers, or BioclimData .
tasmax	Same form as tas: monthly maximum temperature.
tasmin	Same form as tas: monthly minimum temperature.
pr	Same form as tas: monthly precipitation.
window	Integer: length of the rolling window in months (2-11). Default is 3.
na.rm	Logical. If FALSE (default), a single NA makes all outputs NA for that pixel. If TRUE, missing months are skipped.
...	Not currently used.

Value

Named vector (length 19), matrix (pixels x 19), or 19-layer [SpatRaster](#).

bioclim_warnings	<i>Retrieve stored warning messages</i>
------------------	---

Description

Returns the character vector of warning messages currently held in the xbioclim message store. Under normal usage the store is automatically flushed by [check_messages\(\)](#) after every C++ call, but you can inspect it manually before that point if needed.

Usage

```
bioclim_warnings()
```

Value

A character vector (possibly empty).

See Also

[bioclim_errors\(\)](#), [has_warning\(\)](#), [clear_messages\(\)](#)

Examples

```
clear_messages()
bioclim_warnings() # character(0)
```

bioclim_window

Bioclimatic variables over an arbitrary window of months

Description

Computes the 19 standard bioclimatic variables over a user-defined subset of months. This is useful when a species' relevant season does not coincide with fixed quarters (e.g. eBird Status & Trends breeding windows).

Usage

```
bioclim_window(
  tas,
  tasmax,
  tasmin,
  pr,
  months = NULL,
  start = NULL,
  end = NULL,
  window = 3L,
  na.rm = FALSE,
  ...
)
```

Arguments

tas	Numeric vector (length 12), matrix (pixels x 12), SpatRaster with 12 layers, or BioclimData .
tasmax	Same form as tas: monthly maximum temperature.
tasmin	Same form as tas: monthly minimum temperature.
pr	Same form as tas: monthly precipitation.
months	Integer vector of 1-based months in the window, e.g. <code>c(6, 7)</code> for Jun-Jul or <code>c(12, 1, 2)</code> for DJF. If start and end are supplied, months is derived automatically.
start,end	Optional character strings in "MM-DD" format. When both are supplied, the months that overlap the day range are selected. Wrap-around across the year end is supported.
window	Integer: length of the internal rolling sub-window used to compute <code>bio08..bio19</code> . Defaults to 3 (quarterly). Must be $\leq \text{length}(\text{months})$ for the rolling variables to be non-NA.
na.rm	Logical. If FALSE (default), a single NA among the selected months makes all outputs NA for that pixel. If TRUE, missing months are skipped.
...	Not currently used.

Details

The base variables (bio01..bio07, bio12..bio15) are computed directly over the selected months. The rolling variables (bio08..bio11, bio16..bio19) are computed over the best contiguous window-month period that is fully contained in months. If `length(months) < window` or no such contiguous period exists, those columns are NA.

Value

- Vector input: a named numeric vector of length 19 (bio01..bio19).
- Matrix / [BioclimData](#) input: a numeric matrix (pixels x 19).
- [SpatRaster](#) input: a 19-layer [SpatRaster](#).

See Also

[bioclim_rolling\(\)](#) for rolling optimal windows over the full year with a free window length.

BioclimData

Create a BioclimData Object

Description

Constructs a [BioclimData](#) object from monthly climate data. Plain numeric vectors of length 12 are automatically coerced to 1-row matrices so that single-pixel and raster-block inputs are handled uniformly.

Usage

```
BioclimData(tas, tasmax, tasmin, pr)
```

Arguments

tas	Numeric vector (length 12) or matrix (pixels × 12): monthly mean temperature.
tasmax	Numeric vector (length 12) or matrix (pixels × 12): monthly maximum temperature.
tasmin	Numeric vector (length 12) or matrix (pixels × 12): monthly minimum temperature.
pr	Numeric vector (length 12) or matrix (pixels × 12): monthly precipitation.

Value

A [BioclimData](#) object.

Examples

```

tas    <- 1:12
tasmax <- 2:13
tasmin <- 0:11
pr     <- 1:12
bd <- BioclimData(tas, tasmax, tasmin, pr)
bd

```

BioclimData-class

*S4 Class for Bioclimatic Input Data***Description**

Holds monthly climate data for one or more pixels (a raster block) and exposes S4 methods for computing each of the 19 standard bioclimatic variables as well as the full batch computation. Each slot is a numeric matrix with **12 columns** (one per calendar month) and **one row per pixel**; single-pixel inputs (plain numeric vectors of length 12) are automatically promoted to 1-row matrices by the constructor.

The S4 methods delegate to the compiled C++ routines exported by the package (e.g. `bio01_cpp`, `bioclim_cpp`), mirroring the `xbioclim` convention for all 19 variables.

Value

An S4 object of class `BioclimData` holding four numeric matrices (`tas`, `tasmax`, `tasmin`, `pr`), each with 12 columns (one per calendar month) and one row per pixel.

Slots

`tas` Numeric matrix (pixels $\times$ 12): monthly mean temperature.
`tasmax` Numeric matrix (pixels $\times$ 12): monthly maximum temperature.
`tasmin` Numeric matrix (pixels $\times$ 12): monthly minimum temperature.
`pr` Numeric matrix (pixels $\times$ 12): monthly precipitation.

BioclimModel

*Create a BioclimModel Object***Description**

Constructs a [BioclimModel-class](#) S4 object backed by a C++ `BioclimModel` instance. The four monthly climate arrays are validated and passed to the C++ object; all bioclimatic variable computations delegate to that object via `Rcpp`.

Usage

```
BioclimModel(tas, tasmax, tasmin, pr)
```

Arguments

tas	Numeric vector of length 12: monthly mean temperature.
tasmax	Numeric vector of length 12: monthly maximum temperature.
tasmin	Numeric vector of length 12: monthly minimum temperature.
pr	Numeric vector of length 12: monthly precipitation.

Value

A `BioclimModel-class` object.

Examples

```
tas    <- 1:12
tasmax <- 2:13
tasmin <- 0:11
pr     <- 1:12
m <- BioclimModel(tas, tasmax, tasmin, pr)
bio01(m)
bioclim(m)
```

BioclimModel-class	<i>BioclimModel S4 Class</i>
--------------------	------------------------------

Description

An S4 class that wraps a C++ `BioclimModel` object via an opaque external pointer handle, following the terra package pattern for C++ object handles. All 19 bioclimatic variable computations are delegated to the underlying C++ object via `Rcpp`.

Value

An S4 object of class `BioclimModel` wrapping a C++ `BioclimModel` instance via an `externalptr` handle in slot `pntr`.

Slots

`pntr` An `externalptr` to the underlying C++ `BioclimModel` object.

BioclimModel-methods *S4 Methods for BioclimModel Objects*

Description

S4 method implementations for all 19 bioclimatic variable functions and `bioclim` that dispatch to the underlying C++ object via Rcpp when the first argument is a `BioclimModel-class` instance.

Value

For `bio01`–`bio19`: a single numeric value with the corresponding bioclimatic variable computed from the monthly climate data stored in the object. For `bioclim`: a named numeric vector of length 19 (`bio01` through `bio19`).

`clear_messages` *Clear all stored messages*

Description

Discards all error and warning messages currently held in the xbioclim message store. This is called automatically by `check_messages()` after propagating messages to R conditions, but you can call it manually to reset state between operations.

Usage

```
clear_messages()
```

Value

Invisible NULL.

See Also

`bioclim_errors()`, `bioclim_warnings()`

Examples

```
clear_messages()
has_error()      # FALSE
has_warning()    # FALSE
```

ClimateBlock

*C++ ClimateBlock class for batch bioclim computation***Description**

An Rcpp module class exposing the C++ ClimateBlock implementation of the xbioclim library. Accepts four `n_pixels` x 12 matrices of monthly climate data and computes all 19 bioclimatic variables for each pixel via a compiled C++ back-end.

Details

ClimateBlock is loaded into the package namespace when the package is attached (via `loadModule` in `.onLoad`).

Value

`new(ClimateBlock, tas, tasmax, tasmin, pr)` returns an Rcpp module object of class ClimateBlock. Its `$n_pixels()` method returns a single integer and its `$compute()` method returns an `n_pixels` x 19 numeric matrix with columns named `bio01` through `bio19`.

Constructor

`new(ClimateBlock, tas, tasmax, tasmin, pr)`

`tas` Numeric matrix of dimensions `n_pixels` x 12: monthly mean temperature for each pixel.

`tasmax` Numeric matrix of dimensions `n_pixels` x 12: monthly maximum temperature for each pixel.

`tasmin` Numeric matrix of dimensions `n_pixels` x 12: monthly minimum temperature for each pixel.

`pr` Numeric matrix of dimensions `n_pixels` x 12: monthly precipitation for each pixel.

All four matrices must have exactly 12 columns and the same number of rows.

Methods

`n_pixels()` Returns the number of pixels (integer).

`compute()` Computes the 19 bioclimatic variables for all pixels and returns an `n_pixels` x 19 numeric matrix with columns named `bio01` through `bio19`.

Examples

```
tas <- matrix(rep(1:12, 3), nrow = 3, byrow = TRUE)
tasmax <- tas + 1
tasmin <- tas - 1
pr <- matrix(rep(1:12, 3), nrow = 3, byrow = TRUE)
block <- new(ClimateBlock, tas, tasmax, tasmin, pr)
block$n_pixels() # 3
result <- block$compute() # 3 x 19 matrix
```

create_mask

*Create a binary mask raster from a polygon source***Description**

Converts a polygon source to a binary raster mask aligned to a reference raster, and optionally applies that mask to an input raster (replacing pixels outside all polygons with NA).

Usage

```
create_mask(
  polygon,
  reference_raster = NULL,
  output = tempfile(fileext = ".tif"),
  apply_to = NULL,
  apply_output = tempfile(fileext = ".tif")
)
```

Arguments

polygon	Polygon source: an sf object, a character file path to an OGR-readable vector source, or a terra::SpatRaster used directly as a binary mask.
reference_raster	Character string: path to a GDAL-readable raster used to set the spatial reference (extent, resolution, CRS) of the output mask. Ignored when polygon is a SpatRaster.
output	Character string: file path for the output mask GeoTIFF. Defaults to a temporary file.
apply_to	Character string or NULL: if supplied, path to a GDAL-readable raster to which the mask will be applied. Pixels outside all polygons are set to NA in the result.
apply_output	Character string: file path for the masked output raster. Used only when apply_to is not NULL. Defaults to a temporary file.

Details

The function accepts three types of polygon input:

sf object Written to a temporary GeoJSON file via `sf::st_write()` and then rasterized. Requires the **sf** package.

Character file path Passed directly to the C++ rasterizer. Any OGR-readable format is supported (shapefile, GeoJSON, GeoPackage, etc.).

SpatRaster object Used directly as a pre-made mask raster. Must already be binary (0/1) and aligned to reference_raster. Requires the **terra** package.

All GDAL-dependent operations skip gracefully (returning NULL invisibly with a message) when the package was built without GDAL support.

Value

When `apply_to` is `NULL`, returns the path to the binary mask GeoTIFF invisibly. When `apply_to` is provided, returns the path to the masked output raster invisibly.

See Also

[rasterize_mask_cpp\(\)](#), [apply_mask_cpp\(\)](#)

Examples

```
if (has_gdal()) {
  # Requires GDAL support at build time.
  ref <- system.file("extdata", "tiny.tif", package = "xbioclim")
  poly <- tempfile(fileext = ".geojson")
  writeLines(
    paste0(
      '{"type":"FeatureCollection","features":[{"type":"Feature",',
      '"geometry":{"type":"Polygon",',
      '"coordinates":[[[0,0],[1,0],[1,1],[0,1],[0,0]]]},',
      '"properties":{}}]}'
    ),
    poly
  )
  mask_path <- create_mask(poly, ref)
}
```

cuda_device_count

Count available CUDA GPU devices

Description

Returns the number of CUDA-capable GPUs available on this machine. Returns 0 when the package was built without CUDA support or when no CUDA-capable device is found.

Usage

```
cuda_device_count()
```

Value

Non-negative integer: number of CUDA devices detected.

See Also

[has_cuda](#), [cuda_device_info](#)

cuda_device_info	<i>Query properties of the first CUDA GPU device</i>
------------------	--

Description

Returns a named list with hardware information about the first CUDA-capable GPU. Returns an empty list when no CUDA device is available or when the package was built without CUDA.

Usage

```
cuda_device_info()
```

Value

Named list with:

name Character: GPU model name.

memory_gb Numeric: total global memory in gigabytes.

compute_capability Character: e.g. "8.0" for A100.

An empty list when no CUDA device is detected.

See Also

[has_cuda](#), [cuda_device_count](#)

cuda_info	<i>Query CUDA GPU device information</i>
-----------	--

Description

Returns hardware information about the first CUDA-capable GPU detected on this machine.

Usage

```
cuda_info()
```

Value

A named list with name (character GPU model name), memory_gb (numeric total memory in GB), and compute_capability (character, e.g. "8.0" for A100). Returns an empty list when no CUDA device is available.

See Also

[has_cuda](#), [bioclim_engine](#)

Examples

```
cuda_info()
```

engine_compute	<i>Run the bioclimatic-variable computation pipeline</i>
----------------	--

Description

Reads all monthly climate input rasters tile by tile, computes the bioclimatic variables for every pixel, and writes all 19 variables to a single multi-band GeoTIFF named `bio.tif` inside the output directory. Peak memory is proportional to the tile size, not the full raster size.

Usage

```
engine_compute(xptr)
```

Arguments

xptr External pointer returned by [engine_create](#).

Details

Requires GDAL support. Stops with an informative error when the package was built without GDAL.

Value

Character scalar: the output directory path (same as the value passed to [engine_set_output](#)).

See Also

[engine_create](#), [engine_set_output](#), [has_gdal](#)

engine_create	<i>Create a new BioclimEngine instance</i>
---------------	--

Description

Allocates a new BioclimEngine C++ object and returns an opaque external pointer to it. Use the companion `engine_*`() functions to configure and run the engine.

Usage

```
engine_create()
```

Value

An externalptr to a new BioclimEngine object.

See Also

[engine_open](#), [engine_compute](#)

engine_open	<i>Configure monthly climate input files</i>
-------------	--

Description

Associates four sets of raster file paths with the engine. Each vector must contain either one multi-band file (12 bands) or twelve single-band files (one per calendar month).

Usage

```
engine_open(xptr, tas_files, tasmax_files, tasmin_files, pr_files)
```

Arguments

- | | |
|--------------|--|
| xptr | External pointer returned by engine_create . |
| tas_files | Character vector (length 1 or 12): mean temperature. |
| tasmax_files | Character vector (length 1 or 12): maximum temperature. |
| tasmin_files | Character vector (length 1 or 12): minimum temperature. |
| pr_files | Character vector (length 1 or 12): precipitation. |

Value

NULL invisibly.

See Also

[engine_create](#), [engine_compute](#)

engine_set_device	<i>Set the compute device for a BioclimEngine instance</i>
-------------------	--

Description

Controls whether the computation runs on a CUDA GPU or the CPU. When "auto" is selected the engine uses the GPU if at least one CUDA device is present, otherwise it falls back to the CPU. GPU requests on systems without a CUDA device silently fall back to the CPU.

Usage

```
engine_set_device(xptr, device)
```

Arguments

xptr	External pointer returned by engine_create .
device	Character scalar: one of "auto", "cpu", or "gpu".

Value

NULL invisibly.

See Also

[engine_create](#), [has_cuda](#), [bioclim_engine](#)

engine_set_mask	<i>Set an optional mask raster</i>
-----------------	------------------------------------

Description

Pixels where the mask band equals 0 or NaN receive NaN (no-data) in every output band. Pass an empty string to disable masking.

Usage

```
engine_set_mask(xptr, mask_path)
```

Arguments

xptr	External pointer returned by engine_create .
mask_path	Character scalar: mask raster path, or "" for none.

Value

NULL invisibly.

See Also

[engine_create](#), [engine_compute](#)

engine_set_output	<i>Set the output raster path</i>
-------------------	-----------------------------------

Description

The engine will create (or overwrite) a multi-band GeoTIFF named `bio.tif` inside this directory when [engine_compute](#) is called.

Usage

```
engine_set_output(xptr, path)
```

Arguments

xptr	External pointer returned by engine_create .
path	Character scalar: output directory path.

Value

NULL invisibly.

See Also

[engine_create](#), [engine_compute](#)

engine_set_threads	<i>Set the number of OpenMP threads</i>
--------------------	---

Description

Controls the number of threads used in the per-pixel inner loop inside each tile. Values less than 1 are clamped to 1.

Usage

```
engine_set_threads(xptr, n)
```

Arguments

xptr	External pointer returned by engine_create .
n	Integer scalar: number of threads.

Value

NULL invisibly.

See Also

[engine_create](#), [engine_compute](#)

<code>engine_set_variables</code>	<i>Select which bioclimatic variables to write</i>
-----------------------------------	--

Description

Restricts the output to a subset of the 19 standard bioclimatic variables. The engine always computes all 19 internally (they share intermediate values). With the multi-band output file, all 19 bands are written and the [bioclim_engine](#) R wrapper subsets the returned `SpatRaster`.

Usage

```
engine_set_variables(xptr, variables)
```

Arguments

- | | |
|------------------------|--|
| <code>xptr</code> | External pointer returned by engine_create . |
| <code>variables</code> | Integer vector with elements in 1..19. |

Value

NULL invisibly.

See Also

[engine_create](#), [engine_compute](#)

<code>era5_bioclim</code>	<i>Compute Bioclimatic Variables from ERA5-Land Data</i>
---------------------------	--

Description

End-to-end pipeline that reads ERA5-Land hourly GRIB/NetCDF files, aggregates them to CHELSA-compatible monthly climate variables, and computes the 19 standard bioclimatic variables (BIO01–BIO19).

Usage

```
era5_bioclim(
  t2m_files,
  tp_files,
  year,
  output = tempdir(),
  to_celsius = TRUE,
  variables = 1:19,
  ncores = 1L,
  save_monthly = FALSE
)
```

Arguments

t2m_files	Character vector of 12 file paths to monthly ERA5-Land hourly 2-m temperature files (one per calendar month, January–December). Each file may be GRIB or NetCDF.
tp_files	Character vector of 12 file paths to monthly ERA5-Land hourly total precipitation files (same order as t2m_files).
year	Integer: the calendar year (used to determine days per month).
output	Character path to an output directory for GeoTIFF files. Defaults to a temporary directory.
to_celsius	Logical: convert temperatures to Celsius? Default TRUE for WorldClim-convention bioclimatic variables.
variables	Integer vector of bioclimatic variables to compute (1–19). Default 1:19 (all).
ncores	Integer: OpenMP threads for aggregation. Default 1L.
save_monthly	Logical: write intermediate monthly GeoTIFFs? Default FALSE.

Details

The pipeline proceeds in three stages:

1. **Monthly aggregation:** For each of the 12 calendar months, hourly t2m is aggregated to tas, tasmax, and tasmin; hourly tp is summed to pr.
2. **Stack:** The 12 monthly layers are assembled into `terra::SpatRaster` objects with 12 bands each.
3. **Bioclim:** `bioclim_raster` computes BIO01–BIO19.

Value

A `terra::SpatRaster` with one layer per bioclimatic variable.

See Also

[era5_t2m_to_monthly](#), [era5_tp_to_monthly](#), [bioclim_raster](#)

Examples

```
## Not run:
# Paths to ERA5-Land GRIB files on the HPC cluster
t2m_files <- sprintf("era5land_t2m_hourly_2020_%02d.grib", 1:12)
tp_files  <- sprintf("era5land_tp_hourly_2020_%02d.grib", 1:12)

bio <- era5_bioclim(t2m_files, tp_files, year = 2020L, ncores = 4L)
terra::plot(bio[[1]]) # BI001

## End(Not run)
```

era5_bioclim_years	<i>Generate Bioclimatic Variables for Multiple Years</i>
--------------------	--

Description

Batch-processes multiple years of ERA5-Land data through the [era5_bioclim](#) pipeline.

Usage

```
era5_bioclim_years(
  base_dir,
  years,
  output_dir,
  t2m_pattern = "era5land_t2m_hourly_%d_%02d.grib",
  tp_pattern = "era5land_tp_hourly_%d_%02d.grib",
  ...
)
```

Arguments

base_dir	Character: base directory containing ERA5-Land raw data. Expected structure: base_dir/t2m/YYYY/era5land_t2m_hourly_YYYY_MM.grib and base_dir/tp/YYYY/era5land_tp_h
years	Integer vector of years to process.
output_dir	Character: output directory.
t2m_pattern	Character: filename pattern for t2m files. Must contain %d for year and %02d for month. Default: "era5land_t2m_hourly_%d_%02d.grib".
tp_pattern	Character: filename pattern for tp files. Default: "era5land_tp_hourly_%d_%02d.grib".
...	Additional arguments passed to era5_bioclim .

Value

A named list of terra::SpatRaster objects, one per year.

Examples

```
## Not run:
bio_all <- era5_bioclim_years(
  base_dir  = "/scratch/era5-land/raw",
  years     = 1980:2020,
  output_dir = "/scratch/era5-land/bioclim",
  ncores    = 8L
)

## End(Not run)
```

gdal_can_open

Check whether GDAL can open a raster file

Description

A lightweight diagnostic that tries to open the specified path via GDAL and returns TRUE if successful, FALSE if GDAL cannot open it. Stops with an informative error if the package was built without GDAL support.

Usage

```
gdal_can_open(path)
```

Arguments

path Character string: path to the raster file.

Value

Logical TRUE if GDAL can open the file, FALSE otherwise.

Examples

```
if (has_gdal()) {
  gdal_can_open(system.file("extdata", "tiny.tif", package = "xbioclim"))
}
```

gdal_info	<i>Return metadata about a GDAL-readable raster</i>
-----------	---

Description

Opens the raster at path and returns a named list with dimensions, geotransform, coordinate reference system, and per-band scale/offset values.

Usage

```
gdal_info(path)
```

Arguments

path Character string: path to the raster file.

Details

Data values are always returned as double (float64) in memory. If the raster bands carry GDAL scale/offset metadata (e.g. packed integers), those are reported here and applied automatically by `GdalReader::read_window()`.

Stops with an informative error if the package was built without GDAL support.

Value

Named list with the following elements:

path Character: the path as supplied.

nrows Integer: number of rows (Y pixels).

ncols Integer: number of columns (X pixels).

nbands Integer: number of raster bands.

geotransform Numeric vector of length 6 (GDAL convention): [x_origin, pixel_width, rotation_x, y_origin, rotation_y, pixel_height].

crs Character: WKT coordinate reference system string, or an empty string if not defined.

scale Numeric vector (one per band): GDAL scale factor (1.0 if not set).

offset Numeric vector (one per band): GDAL offset (0.0 if not set).

Examples

```
if (has_gdal()) {
  info <- gdal_info(
    system.file("extdata", "tiny.tif", package = "xbioclim")
  )
  str(info)
}
```

has_cuda	<i>Check CUDA GPU availability</i>
----------	------------------------------------

Description

Returns TRUE when at least one CUDA-capable GPU is detected at runtime. Returns FALSE when the package was built without CUDA support or when no CUDA-capable device is present.

Usage

```
has_cuda()
```

Value

Logical scalar: TRUE if at least one CUDA GPU is available.

See Also

[cuda_info](#), [bioclim_engine](#)

Examples

```
has_cuda()
```

has_error	<i>Check whether any errors are stored</i>
-----------	--

Description

Check whether any errors are stored

Usage

```
has_error()
```

Value

TRUE if the message store contains at least one error, FALSE otherwise.

See Also

[bioclim_errors\(\)](#), [has_warning\(\)](#)

Examples

```
clear_messages()
has_error() # FALSE
```

has_gdal	<i>Check whether the package was built with GDAL support</i>
----------	--

Description

Returns TRUE when xbioclim was compiled with GDAL and the native BioclimEngine tiled pipeline is available, FALSE otherwise.

Usage

```
has_gdal()
```

Details

Internally the function calls [gdal_can_open](#) with a dummy path and inspects whether the resulting error message indicates an absent GDAL build.

Value

Logical scalar: TRUE if GDAL is available, FALSE otherwise.

Examples

```
has_gdal()
```

has_warning	<i>Check whether any warnings are stored</i>
-------------	--

Description

Check whether any warnings are stored

Usage

```
has_warning()
```

Value

TRUE if the message store contains at least one warning, FALSE otherwise.

See Also

[bioclim_warnings\(\)](#), [has_error\(\)](#)

Examples

```
clear_messages()
has_warning() # FALSE
```

quarterly_fixed	<i>Fixed-quarter seasonal variables</i>
-----------------	---

Description

Convenience wrapper around [quarterly_variables\(\)](#) for the four standard fixed quarters.

Usage

```
quarterly_fixed(
  tas,
  tasmax,
  tasmin,
  pr,
  quarter,
  type = c("meteorological", "calendar"),
  na.rm = FALSE,
  ...
)
```

Arguments

tas	Numeric vector (length 12), matrix (pixels x 12), SpatRaster with 12 layers, or BioclimData .
tasmax	Same form as tas: monthly maximum temperature.
tasmin	Same form as tas: monthly minimum temperature.
pr	Same form as tas: monthly precipitation.
quarter	Integer 1-4.
type	Character: "meteorological" (DJF, MAM, JJA, SON) or "calendar" (JFM, AMJ, JAS, OND).
na.rm	Logical. If FALSE (default), a single NA among the selected months makes all outputs NA for that pixel. If TRUE, missing months are skipped.
...	Not currently used.

Value

Same structure as [quarterly_variables\(\)](#): a named numeric vector of length 6 for vector input, a numeric matrix (pixels x 6) for matrix or [BioclimData](#) input, or a 6-layer [SpatRaster](#) for [SpatRaster](#) input, with variables tmean_s, tmax_max, tmin_min, trange, pr_tot, pr_cv.

quarterly_rolling	<i>Rolling-quarter seasonal variables</i>
-------------------	---

Description

Convenience wrapper around [quarterly_variables\(\)](#) for a rolling 3-month quarter starting at start.

Usage

```
quarterly_rolling(tas, tasmax, tasmin, pr, start, na.rm = FALSE, ...)
```

Arguments

tas	Numeric vector (length 12), matrix (pixels x 12), SpatRaster with 12 layers, or BioclimData .
tasmax	Same form as tas: monthly maximum temperature.
tasmin	Same form as tas: monthly minimum temperature.
pr	Same form as tas: monthly precipitation.
start	Integer 1-12: starting month.
na.rm	Logical. If FALSE (default), a single NA among the selected months makes all outputs NA for that pixel. If TRUE, missing months are skipped.
...	Not currently used.

Value

Same structure as [quarterly_variables\(\)](#): a named numeric vector of length 6 for vector input, a numeric matrix (pixels x 6) for matrix or [BioclimData](#) input, or a 6-layer [SpatRaster](#) for [SpatRaster](#) input, with variables tmean_s, tmax_max, tmin_min, trange, pr_tot, pr_cv.

quarterly_variables	<i>Quarterly / seasonal climate variables</i>
---------------------	---

Description

Computes six quarterly/seasonal variables for an arbitrary set of months.

Usage

```
quarterly_variables(tas, tasmax, tasmin, pr, months, na.rm = FALSE, ...)
```

```
## Default S3 method:
```

```
quarterly_variables(tas, tasmax, tasmin, pr, months, na.rm = FALSE, ...)
```

Arguments

tas	Numeric vector (length 12), matrix (pixels x 12), SpatRaster with 12 layers, or BioclimData .
tasmax	Same form as tas: monthly maximum temperature.
tasmin	Same form as tas: monthly minimum temperature.
pr	Same form as tas: monthly precipitation.
months	Integer vector of 1-based month indices to include (e.g. c(1, 2, 3) for a fixed quarter, or c(12, 1, 2) for DJF).
na.rm	Logical. If FALSE (default), a single NA among the selected months makes all outputs NA for that pixel. If TRUE, missing months are skipped.
...	Not currently used.

Details

The output variables are:

- tmean_s — mean tas over the selected months.
- tmax_max — maximum monthly tasmax among the selected months.
- tmin_min — minimum monthly tasmin among the selected months.
- trange — tmax_max - tmin_min.
- pr_tot — total pr over the selected months.
- pr_cv — coefficient of variation of monthly pr ($100 * \text{sd_pop} / \text{mean}$), matching the BIO15 scaling.

Use [quarterly_fixed\(\)](#) for the four standard fixed quarters and [quarterly_rolling\(\)](#) for rolling 3-month windows.

Value

- Vector input: a named numeric vector of length 6.
- Matrix / [BioclimData](#) input: a numeric matrix (pixels x 6) with columns tmean_s, tmax_max, tmin_min, trange, pr_tot, pr_cv.
- [SpatRaster](#) input: a 6-layer [SpatRaster](#) with the same geometry and the same names.

rasterize_mask_cpp	<i>Rasterize a vector polygon layer to a binary mask raster</i>
--------------------	---

Description

Burns all polygon features from a vector source into a new single-band GeoTIFF raster that is spatially aligned to a reference raster. Pixels that fall inside at least one polygon are set to 1; all other pixels are set to 0.

Usage

```
rasterize_mask_cpp(vector_path, ref_raster_path, output_mask_path)
```

Arguments

vector_path	Character string: path to any OGR-readable vector source (shapefile, GeoJSON, GeoPackage, etc.).
ref_raster_path	Character string: path to a GDAL-readable raster used as the spatial reference (extent, resolution, CRS).
output_mask_path	Character string: file path where the output GDT_Byte GeoTIFF will be written (created or overwritten).

Details

This function is the low-level C++ entry point. Most users should call the higher-level [create_mask](#) wrapper instead.

Stops with an informative error if the package was built without GDAL support.

Value

Invisibly returns NULL. The side effect is the creation of the output mask raster at output_mask_path.

See Also

[create_mask](#), [apply_mask_cpp](#)

Examples

```
if (has_gdal()) {
  # Requires GDAL support at build time.
  ref <- system.file("extdata", "tiny.tif", package = "xbioclim")
  poly <- tempfile(fileext = ".geojson")
  mask <- tempfile(fileext = ".tif")
  writeLines(
    '{"type": "FeatureCollection", "features": [{"type": "Feature",
      "geometry": {"type": "Polygon", "coordinates": [[[0,0],[1,0],[1,1],[0,1],[0,0]]]},
      "properties": {}}]}' ,
    poly)
  rasterize_mask_cpp(poly, ref, mask)
}
```

Description

xbioclim mirrors the SpatMessages pattern from the terra package to provide a clean mechanism for propagating error and warning messages across the C++/R boundary.

How the cross-boundary workflow works:

1. A C++ routine performs its computation and, instead of throwing directly, records any error or warning strings into a session-level message store.
2. After every `.Call()` invocation, the R-side wrapper calls `check_messages()` to inspect the store and re-raise any messages as native R conditions (`stop()` for errors, `warning()` for warnings).
3. Users can also inspect the store directly with `bioclim_errors()` and `bioclim_warnings()` before R conditions are raised, or clear it with `clear_messages()`.

Functions available to users:

- `bioclim_errors()` – retrieve stored error messages.
- `bioclim_warnings()` – retrieve stored warning messages.
- `has_error()` – TRUE if at least one error is stored.
- `has_warning()` – TRUE if at least one warning is stored.
- `clear_messages()` – discard all stored messages.

Functions used internally (and by future C++ glue code):

- `push_error()` – store an error message.
- `push_warning()` – store a warning message.
- `check_messages()` – raise stored messages as R conditions and clear them.

Value

No return value; this page documents the internal message store. `bioclim_errors()` and `bioclim_warnings()` return character vectors, `has_error()` and `has_warning()` return logicals, and `clear_messages()` returns NULL invisibly.

Index

`apply_mask_cpp`, 4, 38
`apply_mask_cpp()`, 22

`bio01` (bioclim-variables), 5
`bio01`, ANY-method (bioclim-variables), 5
`bio01`, BioclimData-method (bioclim-variables), 5
`bio01`, BioclimModel-method (BioclimModel-methods), 19
`bio02` (bioclim-variables), 5
`bio02`, ANY, ANY-method (bioclim-variables), 5
`bio02`, BioclimData, missing-method (bioclim-variables), 5
`bio02`, BioclimModel, missing-method (BioclimModel-methods), 19
`bio03` (bioclim-variables), 5
`bio03`, ANY, ANY-method (bioclim-variables), 5
`bio03`, BioclimData, missing-method (bioclim-variables), 5
`bio03`, BioclimModel, missing-method (BioclimModel-methods), 19
`bio04` (bioclim-variables), 5
`bio04`, ANY-method (bioclim-variables), 5
`bio04`, BioclimData-method (bioclim-variables), 5
`bio04`, BioclimModel-method (BioclimModel-methods), 19
`bio05` (bioclim-variables), 5
`bio05`, ANY-method (bioclim-variables), 5
`bio05`, BioclimData-method (bioclim-variables), 5
`bio05`, BioclimModel-method (BioclimModel-methods), 19
`bio06` (bioclim-variables), 5
`bio06`, ANY-method (bioclim-variables), 5
`bio06`, BioclimData-method (bioclim-variables), 5
`bio06`, BioclimModel-method (BioclimModel-methods), 19
`bio07` (bioclim-variables), 5
`bio07`, ANY, ANY-method (bioclim-variables), 5
`bio07`, BioclimData, missing-method (bioclim-variables), 5
`bio07`, BioclimModel, missing-method (BioclimModel-methods), 19
`bio08` (bioclim-variables), 5
`bio08`, ANY, ANY-method (bioclim-variables), 5
`bio08`, BioclimData, missing-method (bioclim-variables), 5
`bio08`, BioclimModel, missing-method (BioclimModel-methods), 19
`bio09` (bioclim-variables), 5
`bio09`, ANY, ANY-method (bioclim-variables), 5
`bio09`, BioclimData, missing-method (bioclim-variables), 5
`bio09`, BioclimModel, missing-method (BioclimModel-methods), 19
`bio10` (bioclim-variables), 5
`bio10`, ANY-method (bioclim-variables), 5
`bio10`, BioclimData-method (bioclim-variables), 5
`bio10`, BioclimModel-method (BioclimModel-methods), 19
`bio11` (bioclim-variables), 5
`bio11`, ANY-method (bioclim-variables), 5
`bio11`, BioclimData-method (bioclim-variables), 5
`bio11`, BioclimModel-method (BioclimModel-methods), 19
`bio12` (bioclim-variables), 5
`bio12`, ANY-method (bioclim-variables), 5
`bio12`, BioclimData-method (bioclim-variables), 5

- bio12,BioclimModel-method
(BioclimModel-methods), 19
- bio13(bioclim-variables), 5
- bio13,ANY-method (bioclim-variables), 5
- bio13,BioclimData-method
(bioclim-variables), 5
- bio13,BioclimModel-method
(BioclimModel-methods), 19
- bio14(bioclim-variables), 5
- bio14,ANY-method (bioclim-variables), 5
- bio14,BioclimData-method
(bioclim-variables), 5
- bio14,BioclimModel-method
(BioclimModel-methods), 19
- bio15(bioclim-variables), 5
- bio15,ANY-method (bioclim-variables), 5
- bio15,BioclimData-method
(bioclim-variables), 5
- bio15,BioclimModel-method
(BioclimModel-methods), 19
- bio16(bioclim-variables), 5
- bio16,ANY-method (bioclim-variables), 5
- bio16,BioclimData-method
(bioclim-variables), 5
- bio16,BioclimModel-method
(BioclimModel-methods), 19
- bio17(bioclim-variables), 5
- bio17,ANY-method (bioclim-variables), 5
- bio17,BioclimData-method
(bioclim-variables), 5
- bio17,BioclimModel-method
(BioclimModel-methods), 19
- bio18(bioclim-variables), 5
- bio18,ANY,ANY-method
(bioclim-variables), 5
- bio18,BioclimData,missing-method
(bioclim-variables), 5
- bio18,BioclimModel,missing-method
(BioclimModel-methods), 19
- bio19(bioclim-variables), 5
- bio19,ANY,ANY-method
(bioclim-variables), 5
- bio19,BioclimData,missing-method
(bioclim-variables), 5
- bio19,BioclimModel,missing-method
(BioclimModel-methods), 19
- bioclim, 8
- bioclim(bioclim-variables), 5
- bioclim(), 13
- bioclim,ANY,ANY,ANY,ANY-method
(bioclim-variables), 5
- bioclim,BioclimData,missing,missing,missing-method
(bioclim-variables), 5
- bioclim,BioclimModel,ANY,ANY,ANY-method
(BioclimModel-methods), 19
- bioclim-raster, 5
- bioclim-variables, 5
- bioclim_block, 8
- bioclim_engine, 8, 23, 26, 28, 33
- bioclim_errors, 11
- bioclim_errors(), 3, 14, 19, 33, 39
- bioclim_raster, 10, 12, 29
- bioclim_raster(), 5
- bioclim_rolling, 13
- bioclim_rolling(), 16
- bioclim_warnings, 14
- bioclim_warnings(), 3, 12, 19, 34, 39
- bioclim_window, 15
- BioclimData, 6, 7, 14, 15, 16, 16, 35–37
- BioclimData-class, 17
- BioclimModel, 17
- BioclimModel-class, 18
- BioclimModel-methods, 19
- check_messages(), 3, 11, 14, 19, 39
- clear_messages, 19
- clear_messages(), 3, 12, 14, 39
- ClimateBlock, 20
- create_mask, 4, 5, 21, 38
- cuda_device_count, 22, 23
- cuda_device_info, 22, 23
- cuda_info, 23, 33
- engine_compute, 24, 25, 27, 28
- engine_create, 10, 24, 24–28
- engine_open, 25, 25
- engine_set_device, 26
- engine_set_mask, 26
- engine_set_output, 24, 27
- engine_set_threads, 27
- engine_set_variables, 28
- era5_bioclim, 28, 30
- era5_bioclim(), 3
- era5_bioclim_years, 30
- era5_bioclim_years(), 3
- era5_t2m_to_monthly, 29
- era5_t2m_to_monthly(), 3

era5_to_monthly(), [3](#)
era5_tp_to_monthly, [29](#)
era5_tp_to_monthly(), [3](#)

gdal_can_open, [31](#), [34](#)
gdal_info, [32](#)

has_cuda, [22](#), [23](#), [26](#), [33](#)
has_error, [33](#)
has_error(), [3](#), [12](#), [34](#), [39](#)
has_gdal, [10](#), [24](#), [34](#)
has_warning, [34](#)
has_warning(), [3](#), [14](#), [33](#), [39](#)

loadModule, [20](#)

quarterly_fixed, [35](#)
quarterly_fixed(), [37](#)
quarterly_rolling, [36](#)
quarterly_rolling(), [37](#)
quarterly_variables, [36](#)
quarterly_variables(), [35](#), [36](#)

rasterize_mask_cpp, [4](#), [5](#), [37](#)
rasterize_mask_cpp(), [22](#)

sf::st_write(), [21](#)
SpatRaster, [5](#), [13–16](#), [35–37](#)

xbioclim(xbioclim-package), [3](#)
xbioclim-messages, [39](#)
xbioclim-package, [3](#)