# Package 'barrks'

March 27, 2025

**Type** Package

**Title** Calculate Bark Beetle Phenology Using Different Models

**Version** 1.1.1

**Description** Calculate the bark beetle phenology based on raster data or point-related data. There are multiple models implemented for two bark beetle species. The models can be customized and their submodels (onset of infestation, beetle development, diapause initiation, mortality) can be combined. The following models are available in the package: PHENIPS-Clim (first-time release in this package), PHENIPS (Baier et al. 2007) <doi:10.1016/j.foreco.2007.05.020>, RITY (Ogris et al. 2019) <doi:10.1016/j.ecolmodel.2019.108775>, CHAPY (Ogris et al. 2020) <doi:10.1016/j.ecolmodel.2020.109137>, BSO (Jakoby et al. 2019) <doi:10.1111/gcb.14766>, Lange et al. (2008) <doi:10.1007/978-3-540-85081-6_32>, Jönsson et al. (2011) <doi:10.1007/s10584-011-0038-4>. The package may be expanded by models for other bark beetle species in the future.

**URL** https://jjentschke.github.io/barrks/, https://github.com/jjentschke/barrks/

**BugReports** https://github.com/jjentschke/barrks/issues/

**License** GPL (>= 3)

**Encoding** UTF-8

**RoxygenNote** 7.3.2

**RdMacros** Rdpack

**Imports** dplyr, purrr, terra (>= 1.7-18), Rdpack, base, lubridate, readr, stringr

**Collate** 'appearance.R' 'barrks-package.R' 'bso-phenology-results.R' 'bso-phenology.R' 'utils.R' 'bso-plot.R' 'create-daylengths.R' 'create-events.R' 'create-suntimes.R' 'data.R' 'utils-model-functions.R' 'model.R' 'model-bso.R' 'model-phenips.R' 'model-rity.R' 'model-chapy.R' 'model-combine.R' 'model-joensson.R' 'model-lange.R'

'model-phenips-clim.R' 'phenology-properties.R'
'phenology-results.R' 'phenology.R'
'plot-development-diagram.R' 'stations.R' 'utils-doc.R'
'utils-log.R' 'utils-storage.R'

**VignetteBuilder** knitr

**Depends** R (>= 4.3.0)

**Suggests** devtools, geosphere, graphics, knitr, mnormt, ncdf4, rlang, rmarkdown, suncalc, tidyverse, tinytest, utils, viridisLite

**NeedsCompilation** no

**Author** Jakob Jentschke [aut, cre],
FVA BW, Abt. Waldschutz [cph, fnd]

**Maintainer** Jakob Jentschke <jakob.jentschke@forst.bwl.de>

**Repository** CRAN

**Date/Publication** 2025-03-27 10:40:04 UTC

# Contents

analyse.phenology . . . 3
analyse.phenology.bso . . . 4
barrks_colors . . . 5
barrks_data . . . 6
barrks_labels . . . 7
bso_get_flight_rst . . . 7
bso_get_individuals_rst . . . 8
bso_plot_flight_diagram . . . 9
bso_plot_stage_diagram . . . 10
bso_translate_phenology . . . 12
categorize_generations_rst . . . 13
create_daylength_df . . . 14
create_daylength_rst . . . 15
create_onset . . . 16
create_suntimes_df . . . 17
create_suntimes_rsts . . . 18
get_development_rst . . . 19
get_generations_rst . . . 20
get_input_data . . . 22
get_onset_rst . . . 23
list_models . . . 24
model . . . 25
model.bso.apply . . . 26
model.bso.customize . . . 27
model.chapy.apply . . . 29
model.chapy.customize . . . 31
model.joensson.apply . . . 33
model.joensson.customize . . . 35

model.lange.apply . . . 37
model.lange.customize . . . 38
model.phenips.apply . . . 40
model.phenips.customize . . . 41
model.phenips_clim.apply . . . 44
model.phenips_clim.customize . . . 46
model.rity.apply . . . 49
model.rity.customize . . . 51
model_combine . . . 53
params . . . 54
phenology . . . 55
plot_development_diagram . . . 57
properties . . . 59
save_phenology . . . 60
stations_create . . . 61

**Index** **63**

---

`analyse.phenology` *Analyse a phenology*

---

## Description

Here, all functions are listed that are available to analyse the results of a `phenology()`-call.

## Details

Get phenology properties:

- `prop_dates()`
- `prop_filial_generations()`
- `prop_first_date()`
- `prop_hatched_generations()`
- `prop_last_date()`
- `prop_sister_broods()`
- `prop_stations()`
- `prop_year()`

Get phenology results (raster-based):

- `get_development_rst()`
- `get_diapause_rst()`
- `get_generations_rst()`
- `get_hibernating_generations_rst()`
- `get_mortality_rst()`

- get_onset_rst()

Get phenology results (station-based):

- get_development_df()
- get_diapause_df()
- get_generations_df()
- get_hibernating_generations_df()
- get_mortality_df()
- get_onset_df()

Plot phenology results (station-based):

- plot_development_diagram()

## See Also

analyse.phenology.bso

---

`analyse.phenology.bso` *Analyse a BSO generated phenology*

---

## Description

Here, all functions are listed that are available to analyse the results of a bso_phenology() call.

## Details

Get BSO phenology properties:

- prop_dates()
- prop_filial_generations()
- prop_first_date()
- prop_hatched_generations()
- prop_last_date()
- prop_sister_broods()
- prop_stations()
- prop_year()

Get BSO phenology results (raster-based):

- bso_get_flight_rst()
- bso_get_individuals_rst()

Get BSO phenology results (station-based):

- bso_get_flight_df()
- bso_get_individuals_df()

Plot BSO phenology results (station-based):

- bso_plot_flight_diagram()
- bso_plot_stage_diagram()

## See Also

analyse.phenology

---

barrks_colors *Get* barrks *default color palettes*

---

## Description

Get barrks default color palettes.

## Usage

```
barrks_colors(type = "raster")
```

## Arguments

| | |
|---|---|
| type | Select the desired color palette. There are different variants for particular purposes. Allowed values are 'raster', 'diagram_lines', 'diagram_fill', 'bso_flight' and 'bso_stages'. |

## Value

A character vector of hex colors.

## See Also

barrks_labels()

## Examples

```
colors <- barrks_colors()

# use the colors of 'barrks' for your individual plot...
```

barrks_data *Load sample data*

## Description

The package comes with sample data that allow the application of all models available. The following data sets are available:

## Usage

```
barrks_data(dataset = "raster")
```

## Arguments

| | |
|---|---|
| dataset | Choose the data set that should be returned. |

## Details

- `raster` Contains a list of raster weather datasets for a sample area. The data was taken from Deutscher Wetterdienst (DWD).
- `stations` Contains sample station weather data for some cities in Germany. The data was taken from Deutscher Wetterdienst (DWD). Missing global radiation values were replaced by the mean value of the other stations.
- `station_coords` Contains the coordinates (longitude/latitude) of the stations that are included in the `stations` data set. The data was taken from Deutscher Wetterdienst (DWD).

## Value

The respective data set. Can be a list of SpatRasters (for `dataset = 'raster'`) or a data frame.

## Source

- https://opendata.dwd.de/climate_environment/CDC/grids_germany/daily/hyras_de/
- https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/kl/historical/
- https://opendata.dwd.de/climate_environment/CDC/observations_germany/climate/daily/solar/

## Examples

```
# plot first layer of the minimum temperature of the sample raster data
terra::plot(barrks_data()$tmin[[1]])

# print the first lines of the sample station data
head(barrks_data('stations'), 10)

# print the coordinates of the sample stations
barrks_data('station_coords')
```

## barrks_labels *Get* barrks *default legend labels*

### Description

Get barrks default legend labels.

### Usage

```
barrks_labels(type = "raster")
```

### Arguments

| | |
|---|---|
| type | Select the desired legend labels. There are different variants for particular purposes. Allowed values are 'raster', 'diagram', 'bso_flight' and 'bso_stages'. |

### Value

A character vector of labels.

### See Also

barrks_colors()

### Examples

```
labels <- barrks_labels()

# use the labels of 'barrks' for your individual plot...
```

## bso_get_flight_rst *Get flight of individuals (BSO only)*

### Description

Get the number of individuals that are flying.

### Usage

```
bso_get_flight_rst(pheno, generation, flight = 1, dates = prop_dates(pheno))

bso_get_flight_df(
  pheno,
  generation,
  stations = prop_stations(pheno),
  flight = 1,
  dates = prop_dates(pheno)
)
```

### Arguments

| | |
|---|---|
| pheno | A BSO phenology (see bso_phenology()) |
| generation | Generation of interest. For sister broods, 0.5 should be added. |
| flight | Specifies which flight of the respective generation should be returned. Can be 1 (first flight) or 2 (second flight). |
| dates | Select dates that should be present in the output. |
| stations | Pass a character vector to choose stations assigned to pheno by their names, or pass different stations. See stations_create() for details. |

### Value

- bso_get_flight_rst: a multi-layer SpatRaster.
- bso_get_flight_df: a data frame.

### Examples

```
# This may take a few minutes...

# calculate phenology
p <- bso_phenology('bso', barrks_data('stations'), .quiet = TRUE)

# get the number of individuals of the hibernating generation on their first flight
bso_get_flight_df(p, 0, 'Freiburg', flight = 1)
```

---

## bso_get_individuals_rst

*Get individuals (BSO only)*

---

### Description

Get the number of individuals of a generation that are in a specific development stage.

### Usage

```
bso_get_individuals_rst(
  pheno,
  generation,
  stage = "all",
  dates = prop_dates(pheno)
)

bso_get_individuals_df(
  pheno,
  generation,
  stations = prop_stations(pheno),
```

```
  stage = "all",
  dates = prop_dates(pheno)
)
```

## Arguments

| | |
|---|---|
| pheno | A BSO phenology (see bso_phenology()) |
| generation | Generation of interest. For sister broods, 0.5 should be added. |
| stage | If it is a numeric, the individuals of the slots specified will be retrieved. Otherwise it could be one of the following values: all, egg, larva, pupa, white (egg + larva + pupa), maturation, preflight, reproduction, brown (maturation + preflight + reproduction) |
| dates | Select dates that should be present in the output. |
| stations | Pass a character vector to choose stations assigned to pheno by their names, or pass different stations. See stations_create() for details. |

## Value

- bso_get_individuals_rst(): a multi-layer SpatRaster.
- bso_get_individuals(): a data frame.

## Examples

```
# This may take a few minutes...

# calculate phenology
p <- bso_phenology('bso', barrks_data('stations'), .quiet = TRUE)

# get the number of individuals of the hibernating generation that are waiting to fly
bso_get_individuals_df(p, 0, 'Freiburg', stage = 'preflight')
```

---

# bso_plot_flight_diagram

*Plot a flight diagram (BSO only)*

---

## Description

A flight diagram illustrates the daily share of flying individuals over time.

## Usage

```
bso_plot_flight_diagram(
  .pheno,
  .station = prop_stations(.pheno)[1],
  .colors = barrks_colors("bso_flight"),
  .labels = barrks_labels("bso_flight"),
```

```
  .xlim = NULL,
  .legend = "topright",
  ...
)
```

## Arguments

| | |
|---|---|
| .pheno | A BSO phenology (see bso_phenology()) |
| .station | Pass a character vector to choose a station assigned to .pheno by its name, or pass a different station. See stations_create() for details. |
| .colors, .labels | Vectors of colors/labels starting from the first and the second flght of the hibernating generation followed consecutively by elements for the filial generations (first and second flight). |
| .xlim | Date vector of length to that limits the dates plotted. |
| .legend | Pass FALSE if no legend should be plotted. Otherwise the value will be passed to legend() as first argument. Look there for more information. |
| ... | arguments passed to graphics::barplot(). |

## Value

None

## Examples

```
# This may take a few minutes...

# calculate phenology
p <- bso_phenology('bso', barrks_data('stations'), .quiet = TRUE)

bso_plot_flight_diagram(p)
```

---

# bso_plot_stage_diagram

*Plot a stage diagram (BSO only)*

---

## Description

A stage diagram illustrates the share of individuals that are in a specific developmental stage of a particular generation over time.

## Usage

```
bso_plot_stage_diagram(
  .pheno,
  .station = prop_stations(.pheno)[1],
  .stages = list("white", "brown"),
  .lty = c("dashed", "solid"),
  .lwd = 2,
  .colors = barrks_colors("bso_stages"),
  .labels = barrks_labels("bso_stages"),
  .legend_col = TRUE,
  .legend_lty = TRUE,
  ...
)
```

## Arguments

| | |
|---|---|
| `.pheno` | A BSO phenology (see `bso_phenology()`) |
| `.station` | Pass a character vector to choose a station assigned to `.pheno` by its name, or pass a different station. See `stations_create()` for details. |
| `.stages` | List of stages to plot. Elements will be passed to `bso_get_individuals_df()`. Look there for more information. |
| `.lty`, `.lwd` | Vectors of line types or line widths that are used to plot the different stages. Should have the same length as `.stages` or 1. |
| `.colors`, `.labels` | Vectors of colors/labels starting from the hibernating generation followed consecutively by elements for the filial generations (not including sisterbroods). |
| `.legend_col`, `.legend_lty` | Manipulate the appearance of the legends for colors and line types. Pass `TRUE`/`FALSE` to enable/disable the respective legend. For the customization of the respective legend, a list of parameters for graphics::legend can be passed. |
| `...` | arguments passed to `base::plot()`. |

## Value

None

## Examples

```
# This may take a few minutes...

# calculate phenology
p <- bso_phenology('bso', barrks_data('stations'), .quiet = TRUE)

bso_plot_stage_diagram(p)
```

`bso_translate_phenology` *Translate BSO generated phenology*

## Description

A BSO generated phenology cannot be analysed in the same way as other phenology objects. To be able to use the functions that are available for phenology objects returned by `phenology()`, the BSO generated phenology should be translated.

## Usage

```
bso_translate_phenology(pheno, threshold = 0.1, .quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `pheno` | A BSO phenology (see `bso_phenology()`) |
| `threshold` | Share of individuals that must have reached a specific development in the BSO phenology to account for them in the corresponding standard phenology. |
| `.quiet` | If `TRUE`, messages are suppressed. |

## Value

Returns a standard phenology as a list. Look here to find out how a phenology can be analysed. It is not recommended to access the list elements directly.

## Examples

```
# This may take a few minutes...

# calculate and translate BSO phenology
p <- bso_phenology('bso', barrks_data('stations'), .quiet = TRUE)
pt <- bso_translate_phenology(p, .quiet = TRUE)

# print the generations data frame of station 'Freiburg'
df <- get_generations_df(pt, 'Freiburg')
df
```

`categorize_generations_rst` *Make a numeric generations raster categorical*

## Description

Make a numeric generations raster categorical. Useful when mathematical operations were performed with generations rasters (use `get_generations_rst(..., categorical = FALSE)` to get numeric generations rasters).

## Usage

```
categorize_generations_rst(
  rst,
  colors = barrks_colors(),
  labels = barrks_labels()
)
```

## Arguments

| | |
|---|---|
| `rst` | A numeric SpatRaster that represents bark beetle generations. Sister broods are defined by adding `0.5` to the respective generation. |
| `colors, labels` | Vectors of colors/labels starting from zero generations followed consecutively by elements for the respective generations (including sister broods). |

## Value

A categorical SpatRaster.

## Examples

```
# calculate phenology with different models
p1 <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)
p2 <- phenology('phenips', barrks_data(), .quiet = TRUE)

# get the generation as numerical rasters to allow mathematical operations
gens1 <- get_generations_rst(p1, categorical = FALSE)
gens2 <- get_generations_rst(p2, categorical = FALSE)

# calculate the maximum generations from the 2 models
gens_max <- max(gens1, gens2)
# categorize the results
gens_max_cat <- categorize_generations_rst(gens_max)

# plot the categorized raster
terra::plot(gens_max_cat)
# plot the uncategorized raster
terra::plot(gens_max)
```

`create_daylength_df` *Create a data frame of day lengths*

## Description

Generate a data frame of day lengths for given latitudes. The package `geosphere` is required to use this function.

## Usage

```
create_daylength_df(lat, dates, .quiet = FALSE)
```

## Arguments

| | |
|---|---|
| `lat` | Data frame with the fields `station` and `lat`. Defines the latitude for the respective stations. |
| `dates` | Dates that should be processed. |
| `.quiet` | If `TRUE`, messages are suppressed. |

## Value

A data frame with the columns `date`, `station` and `daylength`.

## See Also

`create_daylength_rst()`

## Examples

```
# dates of interest
date_start <- as.Date('2020-01-01')
date_end <- as.Date('2020-12-31')


# calculate day length
dl <- create_daylength_df(barrks_data('station_coords'),
                          seq(date_start, date_end, by = 'day'),
                          .quiet = TRUE)

# print day lengths of station 'Freiburg'
dl[dl$station == 'Freiburg',]
```

`create_daylength_rst` *Create day length rasters*

## Description

Generate a multi-layer SpatRaster of day lengths for a given template. The package `geosphere` is required to use this function.

## Usage

```
create_daylength_rst(
  template,
  dates = terra::time(template),
  crs = "EPSG:4258",
  .quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `template` | (Multi-layer) SpatRaster that determines the spatial extent of the result. |
| `dates` | Dates that should be processed. If not specified, the dates of `template` are used through `terra::time()`. |
| `crs` | Coordinate reference system with longitude/latitude metrics. It is used to project the raster coordinates to be able to retrieve the latitude. |
| `.quiet` | If `TRUE`, messages are suppressed. |

## Value

A multi-layer SpatRaster. Each layer represents one date.

## See Also

`create_daylength_df()`

## Examples

```
# calculate day length, use barrks_data()$tmin as template
dl <- create_daylength_rst(barrks_data()$tmin, .quiet = TRUE)

# plot day length on May 1st, 2015
terra::plot(dl[[terra::time(dl) == '2015-05-01']])
```

create_onset *Create phenological events (onset/diapause/mortality)*

## Description

Generate onset, diapause or mortality manually to be able to run phenology() with observed or arbitrary inputs.

## Usage

```
create_onset(
  template,
  doys = NULL,
  stations = NULL,
  dates = NULL,
  .quiet = FALSE
)

create_diapause(
  template,
  doys = NULL,
  stations = NULL,
  dates = NULL,
  .quiet = FALSE
)

create_mortality(
  template,
  doys = NULL,
  stations = NULL,
  dates = NULL,
  .quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| template | SpatRaster or data frame that determines the spatial and temporal extent of the result. If a single-layer SpatRaster was passed, the temporal extent should be defined by using dates. |
| doys | Numeric vector, (multi-layer) SpatRaster or data frame that specifies the days of year when the event is triggered. Vectors will define the events globally whereas SpatRasters allow spatially explict definitions. For the creation of events based on stations, data frames are used. In that case, the field station specifies the station name and doy indicates the respective day of year. |
| stations | If template is a SpatRaster and doys is a data frame, stations should be passed to define which cells are affected. See stations_create() for details. |

| | |
|---|---|
| dates | Dates to define the temporal extent of the output if template is a single-layer SpatRaster. |
| .quiet | If TRUE, messages are suppressed. |

## Value

A logical multi-layer SpatRaster. Each layer represents one date.

## Functions

- create_onset(): Create a onset.
- create_diapause(): Create a diapause.
- create_mortality(): Create mortality events.

## Examples

```
# load sample data
d <- barrks_data('stations')

# create onset, diapause, mortality
on <- create_onset(d, lubridate::yday('2015-04-15'))
dia <- create_diapause(d, lubridate::yday('2015-08-15'))
mort <- create_mortality(d, lubridate::yday('2015-11-15'))

# claculate phenologe
p <- phenology('phenips-clim', d, .quiet = TRUE,
               .onset = on, .diapause = dia, .mortality = mort)

# plot development
plot_development_diagram(p, .lwd = 4)
```

---

# create_suntimes_df *Create a data frame of sunrises and sunsets*

---

## Description

Generate a data frame that specifies sunrises and sunsets for different coordinates and dates. The package suncalc is required to use this function.

## Usage

```
create_suntimes_df(coords, dates, tz = Sys.timezone(), .quiet = FALSE)
```

### Arguments

| | |
|---|---|
| coords | Data frame with the fields station, lat and lon. Defines the latitude and longitude for the respective stations. |
| dates | Dates that should be processed. |
| tz | Timezone of the results. |
| .quiet | If TRUE, messages are suppressed. |

### Value

A data frame with the columns date, station and sunrise and sunset. The values of sunrise and sunset indicate the respective time in minutes.

### See Also

create_suntimes_rsts()

### Examples

```
date_start <- as.Date('2020-01-01')
date_end <- as.Date('2020-12-31')

st <- create_suntimes_df(barrks_data('station_coords'),
                         seq(date_start, date_end, by = 'day'),
                         .quiet = TRUE)

# print results of station 'Freiburg'
st[st$station == 'Freiburg',]
```

---

## create_suntimes_rsts *Create rasters that indicate sunrise and sunset*

---

### Description

Generate a list of two multi-layer SpatRasters for a given template that indicate sunrise and sunset for the respective cells. The package suncalc is required to use this function.

### Usage

```
create_suntimes_rsts(
  template,
  dates = terra::time(template),
  crs = "EPSG:4258",
  tz = Sys.timezone(),
  .quiet = FALSE
)
```

### Arguments

| | |
|---|---|
| template | (Multi-layer) SpatRaster that determines the spatial extent of the result. |
| dates | Dates that should be processed. If not specified, the dates of template are used through terra::time(). |
| crs | Coordinate reference system with longitude/latitude metrics. It is used to project the raster coordinates to be able to retrieve longitude and latitude. |
| tz | Timezone of the results. |
| .quiet | If TRUE, messages are suppressed. |

### Value

A list with the elements sunrise and sunset which are both multi-layer SpatRasters. The values indicate the respective time in minutes. Each layer represents one date.

### See Also

create_suntimes_df()

### Examples

```
# calculate suntimes, use barrks_data()$tmin as template
st <- create_suntimes_rsts(barrks_data()[[1]], .quiet = TRUE)

# plot results on May 1st, 2015
terra::plot(st$sunrise[[terra::time(st$sunrise) == '2015-05-01']])
terra::plot(st$sunset[[terra::time(st$sunset) == '2015-05-01']])
```

---

## get_development_rst *Get the beetles development*

---

### Description

Get the beetles development of specific generations. A value of -1 implies that the generation is not present yet.

### Usage

```
get_development_rst(pheno, generation, dates = prop_dates(pheno))

get_development_df(
  pheno,
  stations = prop_stations(pheno),
  generation = prop_hatched_generations(pheno),
  dates = prop_dates(pheno)
)
```

### Arguments

| | |
|---|---|
| pheno | A phenology (see phenology()) |
| generation | Generation of interest. For sister broods, 0.5 should be added. get_development_df() allows multiple generations here. |
| dates | Select dates that should be present in the output. |
| stations | Pass a character vector to choose stations assigned to pheno by their names, or pass different stations. See stations_create() for details. |

### Value

- get_development_rst(): A multi-layer SpatRaster.
- get_development_df(): A data frame which contains a field for each generation (gen_1, gen_1.5, gen_2, gen_2.5, ...) requested.

### Examples

```
# calculate station-based phenology
p <- phenology('phenips-clim', barrks_data('stations'), .quiet = TRUE)

# print the development data frame of station 'Freiburg'
df <- get_development_df(p, 'Freiburg')
df[,4:ncol(df)] <- round(df[,4:ncol(df)], 3) # round results
df
```

---

## get_generations_rst *Get generations*

---

### Description

Find out how many generations are present (or have reached a development threshold).

### Usage

```
get_generations_rst(
  pheno,
  dates = prop_last_date(pheno),
  threshold = 0,
  generations = prop_hatched_generations(pheno),
  categorical = TRUE,
  colors = barrks_colors("raster"),
  labels = barrks_labels("raster")
)

get_generations_df(
  pheno,
```

```
  stations = prop_stations(pheno),
  dates = prop_dates(pheno),
  threshold = 0,
  generations = prop_hatched_generations(pheno)
)

get_hibernating_generations_rst(
  pheno,
  categorical = TRUE,
  colors = barrks_colors("raster"),
  labels = barrks_labels("raster")
)

get_hibernating_generations_df(pheno, stations = prop_stations(pheno))
```

## Arguments

| | |
|---|---|
| pheno | A phenology (see phenology()) |
| dates | Select dates that should be present in the output. |
| threshold | Threshold of the beetle development to account for a generation. |
| generations | Numeric vector that determines which generations should be included in the result. |
| categorical | Set FALSE if the SpatRaster that is returned should be numeric. Otherwise, it will be categorical. |
| colors, labels | Vectors of colors/labels starting from zero generations followed consecutively by elements for the respective generations (including sister broods). |
| stations | Pass a character vector to choose stations assigned to pheno by their names, or pass different stations. See stations_create() for details. |

## Value

- get_generations_rst(): A multi-layer SpatRaster.
- get_hibernating_generations_rst(): A SpatRaster. Only available if a mortality event has occured since the diapause started or the model's end date has been reached. Otherwise, the values will be NA.
- get_generations_df(): A data frame.
- get_hibernating_generations_df(): A data frame. Only available if a mortality event has occured since the diapause started or the model's end date has been reached. Otherwise, the values will be NA.

## Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)

# get the generations raster
gens <- get_generations_rst(p)
```

```
# plot the generations raster
terra::plot(gens)
```

---

`get_input_data` *Get (preprocessed) input data*

---

## Description

The function returns a list that contains the input data of the phenology as well as some intermediate results that are needed as preprocessed inputs for the model. The result can be used as input for `phenology()` to avoid redundant calculations.

## Usage

```
get_input_data(pheno)
```

## Arguments

| | |
|---|---|
| pheno | A phenology (see `phenology()`) |

## Value

A list of SpatRasters.

## Examples

```
# setup phenology
p <- phenology('phenips-clim', barrks_data(), .setup_only = TRUE, .quiet = TRUE)

# get the (preprocessed) input data
inputs <- get_input_data(p)

# print the names to show which input data is available
names(inputs)
```

get_onset_rst *Get onset, diapause or mortality*

## Description

Get onset, diapause or mortality as day of year or raw output. Note that multiple mortality events are possible over the season.

## Usage

```
get_onset_rst(pheno, as_doy = TRUE, dates = prop_dates(pheno))

get_onset_df(
  pheno,
  stations = prop_stations(pheno),
  as_doy = TRUE,
  dates = prop_dates(pheno)
)

get_diapause_rst(pheno, as_doy = TRUE, dates = prop_dates(pheno))

get_diapause_df(
  pheno,
  stations = prop_stations(pheno),
  as_doy = TRUE,
  dates = prop_dates(pheno)
)

get_mortality_rst(pheno, as_doy = TRUE, dates = prop_dates(pheno))

get_mortality_df(
  pheno,
  stations = prop_stations(pheno),
  as_doy = TRUE,
  dates = prop_dates(pheno)
)
```

## Arguments

| | |
|---|---|
| pheno | A phenology (see phenology()) |
| as_doy | If TRUE, the day(s) of year will be returned. If FALSE the phenological events will be returned in a raw format. Then, the return value could be used as input for phenology()/bso_phenology() (parameters .onset, .diapause and .mortality). |
| dates | Select dates that should be present in the output. |
| stations | Pass a character vector to choose stations assigned to pheno by their names, or pass different stations. See stations_create() for details. |

## Value

- `get_onset_rst()`, `get_diapause_rst()`, `get_mortality_rst()`: A (multi-layer) SpatRaster.
- `get_onset_df()`, `get_diapause_df()`, `get_mortality_df()`: A data frame.

## Functions

- `get_onset_rst()`: Returns a (multi-layer) SpatRaster of the onset.
- `get_onset_df()`: Returns a data frame of the onset.
- `get_diapause_rst()`: Returns a (multi-layer) SpatRaster of the diapause.
- `get_diapause_df()`: Returns a data frame of the diapause.
- `get_mortality_rst()`: Returns a (multi-layer) SpatRaster of the mortality.
- `get_mortality_df()`: Returns a data frame of the mortality.

## Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)

# plot onset, diapause, mortality
get_onset_rst(p) |> terra::plot()
get_diapause_rst(p) |> terra::plot()
get_mortality_rst(p)[[1]] |> terra::plot()
```

---

# list_models *List all models*

---

## Description

Get the names of all available models.

## Usage

```
list_models()
```

## Value

A character vector.

## Examples

```
# print all available models
list_models()
```

`model` *Get a phenology model*

## Description

Returns a (customized) phenology model.

## Usage

```
model(m, ...)
```

## Arguments

| | |
|---|---|
| `m` | Name of the model or the return value of another `model()`-call. |
| `...` | List of parameters to customize the model. |

## Value

A phenology model. Can be passed to `phenology()`.

## See Also

Look at the customization manuals, to find out which parameters can be customized for a specific model: `model.bso.customize`, `model.phenips.customize`, `model.rity.customize`, `model.chapy.customize`, `model.joensson.customize`, `model.lange.customize`, `model.phenips_clim.customize`.

## Examples

```
# customize the temperature beetles need to fly for PHENIPS-Clim
m <- model('phenips-clim', tfly = 16)

# calculate phenology
p <- phenology(m, barrks_data(), .quiet = TRUE)

# plot generations
gens <- get_generations_rst(p)
terra::plot(gens)
```

`model.bso.apply` *Use BSO*

## Description

This page describes the usage of BSO with `phenology()`. The model-specific inputs are listed and its basic functionality is explained. BSO was published by Jakoby et al. (2019) and parametrized for *Ips typographus* in Switzerland. Note that the onset and the development submodel do not support the usage of a storage (except for some precalculations).

## Arguments

| | |
|---|---|
| `...` | See `phenology()` for a detailled description of the function. |
| `tmin, tmax` | Daily minimum/maximum air temperatures in °C. |
| `sunrise, sunset` | Time of sunrise/sunset in minutes from midnight. Can be created with `create_suntimes_rsts()` or `create_suntimes_df()`. |
| `n` | number of individuals to simulate. |
| `max_generations` | maximum number of generations to calculate. |

## Details

In `barrks`, `phenology()` is used to apply a model. The following code illustrates which inputs are required to apply BSO and which additional parameters are available.

```
bso_phenology("bso", ..., tmin, tmax, sunrise, sunset,
              n = 1e+09, max_generations = 4)

# calculate submodels separately
bso_phenology("bso", ..., .submodels = 'onset',
              tmin, tmax, sunrise, sunset, n = 1e+09)
bso_phenology("bso", ..., .submodels = 'diapause', tmin, tmax)
bso_phenology("bso", ..., .submodels = 'development',
              .onset, .diapause = NULL, .mortality = NULL,
              tmin, tmax, sunrise, sunset,
              max_generations = 4)
```

## Value

The function returns a BSO phenology. Look here to find out how it can be analysed.

## Functioning of the BSO

In the following, the basic functioning of BSO is explained.

- **Onset**: The onset of swarming will start when the degree days of the mean temperature reach a specific threshold and regeneration feeding of the individuals has finished (Look at development for details).
- **Development**: The development of single individuals is simulated. The simulation of each individual is realized by passing a multitude of slots that are grouped in stages. The hourly probability for an individual to enter the next slot depends on the current stage and the phloem temperature. The hourly temperature is derived from the minimum and maximum temperatures using a sine interpolition. The hourly phloem temperature is calculated using Newton's Law of Cooling (see Trân et al. 2007).
- **Diapause**: Specific photoperiod-related dates define when the diapause is initiated at the earliest and at the latest. In between these dates, the diapause is initiated when the mean temperature falls below a specific threshold.
- **Mortality**: BSO does not have a mortality submodel implemented.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

## References

Jakoby O, Lischke H, Wermelinger B (2019). "Climate change alters elevational phenology patterns of the European spruce bark beetle (*Ips typographus*)." *Global Change Biology*, **25**(12), 4048-4063. doi:10.1111/gcb.14766.

Trân JK, Ylioja T, Billings RF, Régnière J, Ayres MP (2007). "Impact of minimum winter temperatures on the population dynamics of *Dendroctonus frontalis*." *Ecological Applications*, **17**(3), 882-899. doi:10.1890/060512.

## See Also

`model()`, `bso_phenology()`, `model.bso.customize`

Other phenology applications: `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips.apply`, `model.phenips_clim.apply`, `model.rity.apply`

---

`model.bso.customize` *Customize BSO*

---

## Description

This page describes the parameters that can be used to customize BSO. The model was developed by Jakoby et al. (2019). Look here to find out how to apply the model.

## Arguments

| | |
|---|---|
| `dd_onset_start_date` | The date, when the degree days start to sum up ('MM-DD'). |
| `dd_onset_base` | Base temperature to calculate degree days to trigger the onset. |

`dd_onset_threshold`
: Degree days that are required before the individuals start regeneration feeding in `slot_dia` of the maturation stage. When the regeneration feeding has finished, the onset is triggered.

`slot_dia`
: Maturation feeding slot where the individuals start regeneration feeding after diapause.

`k`
: Factor for the calculation of the phloem temperature.

`alpha`, `tlo`, `tup`
: Parameters used to calculate the transition probabilities for each stage (except preflight) in the following order: development, maturation feeding, reproduction.

`tfly_min`, `tfly_max`, `pfly_max`, `beta`
: Parameters used to calculate the transition probabilities for the preflight stage.

`num_slots`
: Named vector that defines the number of slots for each stage. The development stage is subdivided into the stages `egg`, `larva` and `pupa`.

`psis`
: Probability that a sister brood will be established.

`slot_sis`
: Maturation feeding slot where the individuals start regeneration feeding before they establish a sister brood.

`model_end_date`
: Date when the model ends (no further development will be modeled).

`diapause_first`
: The day of year when the diapause could start at the earliest.

`diapause_last`
: The day of year when the diapause could start at the latest.

`tdia_min`
: The diapause will be initiated when the average daily temperature falls below that value.

## Details

In `barrks`, `model()` is used to customize a model. The following code illustrates which parameters are available for BSO and specifies their default values.

```
model("bso",

      # ==== onset ====

      dd_onset_start_date = '01-01',
      dd_onset_base = 5.124198,
      dd_onset_threshold = 100,
      slot_dia = 6,

      # ==== onset + development ====

      k = 2.853738e-02,
      alpha = c(2.549060e-05, 0.0000789, 1.009450e-05),
      tlo = c(-1.297644e+01, 4.760089e+00, -4.424628e+00),
      tup = c(3.600070e+01, 4.002483e+01, 3.999390e+01),
      tfly_min = 16.1064,
      tfly_max = 31.2901,
      pfly_max = 9.863263e-03,
```

```
        beta = 1.363763,

        num_slots = c(
          'reproduction' = 11,
          'egg' = 18,
          'larva' = 45,
          'pupa' = 8,
          'maturation' = 8,
          'preflight' = 1
        ),

        # ==== development ====

        psis = 2.994450e-01,
        slot_sis = 4,

        model_end_date = '12-30',

        # ==== diapause ====

        diapause_first = 210,
        diapause_last = 232,
        tdia_min = 1.645209e+01
)
```

## References

Jakoby O, Lischke H, Wermelinger B (2019). "Climate change alters elevational phenology patterns of the European spruce bark beetle (*Ips typographus*)." *Global Change Biology*, **25**(12), 4048-4063. doi:10.1111/gcb.14766.

## See Also

`model()`, `phenology()`, `model.bso.apply`

Other model customizations: `model.chapy.customize`, `model.joensson.customize`, `model.lange.customize`, `model.phenips.customize`, `model.phenips_clim.customize`, `model.rity.customize`

---

`model.chapy.apply` *Use CHAPY*

---

## Description

This page describes the usage of CHAPY with `phenology()`. The model specific inputs are listed and its basic functionality is explained. CHAPY was published by Ogris et al. (2020) and parametrized for *Pityogenes chalcographus* in Slovenia.

## Arguments

`tmin`, `tmean`, `tmax`
: Daily minimum/mean/maximum air temperatures in °C. For the `development` submodel, the parameter that is obligatory depends on `mode`.

`daylength`
: Length of the day in hours. Can be created with `create_daylength_rst()` or `create_daylength_rst()`.

`mode`
: Specifies which temperature should be used to calculate the development. Can be `min`, `mean` or `max`.

`.submodels`, `.onset`, `.diapause`, `.mortality`, `...`
: See `phenology()` for a detailled description of the function.

## Details

In `barrks`, `phenology()` is used to apply a model. The following code illustrates which inputs are required to apply CHAPY and which additional parameters are available.

```
phenology("chapy", ..., tmin = NULL, tmean = NULL, tmax, daylength, mode = 'max')

# calculate submodels separately
phenology("chapy", ..., .submodels = 'onset', tmax)
phenology("chapy", ..., .submodels = 'diapause', daylength)
phenology("chapy", ..., .submodels = 'mortality', tmax)
phenology("chapy", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmin = NULL, tmean = NULL, tmax = NULL, mode = 'max')
```

## Functioning

The functioning of CHAPY is identical to RITY but it is has a different parametrization.

## References

Ogris N, Ferlan M, Hauptman T, Pavlin R, Kavčič A, Jurc M, de Groot M (2020). "Sensitivity analysis, calibration and validation of a phenology model for *Pityogenes chalcographus* (CHAPY)." *Ecological Modelling*, **430**, 109137. ISSN 0304-3800, doi:10.1016/j.ecolmodel.2020.109137.

## See Also

`model()`, `phenology()`, `model.chapy.customize`

Other phenology applications: `model.bso.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips.apply`, `model.phenips_clim.apply`, `model.rity.apply`

model.chapy.customize *Customize CHAPY*

## Description

This page describes the parameters that can be used to customize CHAPY. The model was developed by Ogris et al. (2020). Look here to find out how to apply the model.

## Arguments

`dd_onset_start_date`
: The date, when the degree days start to sum up ('MM-DD').

`dd_onset_base`
: Base temperature to calculate degree days to trigger the onset.

`dd_onset_threshold`
: Degree days that are required to trigger the onset of infestation. Additionally, the maximum temperature must exceed `tfly`.

`tfly`
: Minimum temperature that beetles need to fly.

`dd_development_base`
: Base temperature to calculate degree days for development.

`dd_total_dev`
: Degree days that are required for a generation to fully develop

`dev_start`, `dev_end`
: Share in total development when the egg development starts and the juvenile beetle's development ends respectively. Usable if the development below/above these thresholds should account for mating, oviposition etc.

`dev_sister_brood`
: Share in the total development, when a sister brood will be established.

`dev_mortal_min`, `dev_mortal_max`
: The beetles are considered to be in white stages (egg, larva, pupa) if their development exceeds `dev_mortal_min` and subceeds `dev_mortal_max`. During these stages, the beetles could die due to a mortality event. `NULL` means that no lower/upper threshold is defined.

`func_ftmin`, `func_ftmean`, `func_ftmax`
: Functions to caclulate the air temperature in forest stands (see Ogris et al. 2019, equations 1 - 3). Each parameter will be passed as SpatRaster:

  - `tmin`: min air temperature
  - `tmean`: mean air temperature
  - `tmax`: maximum air temperature

`func_btmin`, `func_btmean`, `func_btmax`
: Functions to caclulate the bark temperature (see Ogris et al. 2019, equations 4 - 6). Each parameter will be passed as SpatRaster:

  - `ftmin`: min air temperature in forest stands
  - `ftmean`: mean air temperature in forest stands
  - `ftmax`: maximum air temperature in forest stands

`dt_low`, `dt_up`, `topt`, `tmax`, `alpha`, `beta`, `gamma`
: Parameters to calculate the effective bark temperature (see Ogris et al. 2020, equations A.7 - A.9).

`model_end_date`
: Date when the model ends (no further development will be modeled).

`daylength_dia`
: When the daylength falls below this threshold, diapause will be initiated.

`mortality_date`
: Date when all white stages (egg, larva, pupa) die.

## Details

In `barrks`, `model()` is used to customize a model. The following code illustrates which parameters are available for CHAPY and specifies their default values.

```
model("chapy",

      # ==== onset ====

      dd_onset_start_date = '03-09',
      dd_onset_base = 7.4,
      dd_onset_threshold = 216.5,

      # ==== onset + development ====

      tfly = 15.6,

      # ==== development ====

      dd_development_base = 7.4,
      dd_total_dev = 635.4,
      dev_start = 0,
      dev_end = 1,
      dev_sister_brood = 0.5,
      dev_mortal_min = NULL,
      dev_mortal_max = 0.8,

      func_ftmin = function(tmin) { 1.44 + 0.82 * tmin },
      func_ftmean = function(tmean) { 0.50 + 0.81 * tmean },
      func_ftmax = function(tmax) { 1.03 + 0.86 * tmax },

      func_btmin = function(atmin) { 0.56 + 0.99 * atmin },
      func_btmean = function(atmean) { -0.48 + 1.03 * atmean },
      func_btmax = function(atmax) { 0.03 + 0.99 * atmax },

      dt_low = 7.4,
      dt_up = 39.4,
      topt = 30,
      tmax = 41.97,
      alpha = 0.031,
      beta = 5.3,
```

```
        gamma = 1.25,

        model_end_date = '12-31',

        # ==== diapause ====

        daylength_dia = 13.6,

        # ==== mortality ====

        mortality_date = '12-31'
)
```

## References

Ogris N, Ferlan M, Hauptman T, Pavlin R, Kavčič A, Jurc M, De Groot M (2019). "RITY–A phenology model of Ips typographus as a tool for optimization of its monitoring." *Ecological Modelling*, **410**, 108775. doi:10.1016/j.ecolmodel.2019.108775.

Ogris N, Ferlan M, Hauptman T, Pavlin R, Kavčič A, Jurc M, de Groot M (2020). "Sensitivity analysis, calibration and validation of a phenology model for *Pityogenes chalcographus* (CHAPY)." *Ecological Modelling*, **430**, 109137. ISSN 0304-3800, doi:10.1016/j.ecolmodel.2020.109137.

## See Also

`model()`, `phenology()`, `model.chapy.apply`

Other model customizations: `model.bso.customize`, `model.joensson.customize`, `model.lange.customize`, `model.phenips.customize`, `model.phenips_clim.customize`, `model.rity.customize`

---

# `model.joensson.apply` *Use the Jönnson model*

---

## Description

This page describes the usage of the the Jönsson model with `phenology()`. The model specific inputs are listed and its basic functionality is explained. The Jönsson model was published by Jönsson et al. (2011) and parametrized for *Ips typographus* in southern Sweden.

## Arguments

| | |
|---|---|
| `...` | See `phenology()` for a detailled description of the function. |
| `tmean, tmax` | Daily mean/maximum temperatures in °C. |
| `daylength` | Length of the day in hours. Can be created with `create_daylength_rst()` or `create_daylength_rst()`. |
| `mode` | Can be `'fast'` (default) or `'slow'`. Determines if the lower (`'fast'`) or upper (`'slow'`) limit for the development of genertion should be used. |

## Details

In barrks, phenology() is used to apply a model. The following code illustrates which inputs are required to apply the Jönsson model and which additional parameters are available.

```
phenology("joensson", ..., tmean, tmax, daylength, mode = 'fast')

# calculate submodels separately
phenology("joensson", ..., .submodels = 'onset', tmax)
phenology("joensson", ..., .submodels = 'diapause', tmax, daylength)
phenology("joensson", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmean, mode = 'fast')
```

## Value

The function returns a phenology. Look here to find out how it can be analysed.

## Functioning

In the following, the basic functioning of the Jönsson model is explained.

- **Onset**: The onset of swarming is triggerd when the degree days of the maximum temperature reach a specific threshold and the maximum temperature exceeds the minimum flight temperature. The onset of infestation is triggered seven days later to account for a pre-oviposition period.
- **Development**: The development progresses proportional to the degree days of the mean temperature. To account for varying sun exposures, two different thermal thresholds are defined that reflect the lower and the upper limit of development. A generations starts swarming when it has finished its development and the maximum temperature exceeds the minimum flight temperature. Seven days later, the development of a new generation starts.
- **Diapause**: The diapause is initiated when the daylength falls below a threshold. It is recommended to adjust the daylength threshold when applying elsewhere (e.g. values from literature). Jönsson et al. (2011) proposes a model to calculate the daylength threshold based on long-term climate data.
- **Mortality**: The Jönsson model does not have a mortality submodel implemented.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

## References

Jönsson AM, Harding S, Krokene P, Lange H, Åke Lindelöw A, Økland B, Ravn HP, Schroeder LM (2011). "Modelling the potential impact of global warming on *Ips typographus* voltinism and reproductive diapause." *Climatic Change*, **109**, 695-718. doi:10.1007/s1058401100384.

## See Also

`model()`, `phenology()`, `model.joensson.customize`

Other phenology applications: `model.bso.apply`, `model.chapy.apply`, `model.lange.apply`, `model.phenips.apply`, `model.phenips_clim.apply`, `model.rity.apply`

---

`model.joensson.customize`

*Customize the Jönsson model*

---

## Description

This page describes the parameters that can be used to customize the Jönsson model. The model was developed by Jönsson et al. (2011). Look here to find out how to apply the model.

## Arguments

`dd_onset_start_date`
: The date, when the degree days start to sum up ('MM-DD').

`dd_onset_base`
: Base temperature to calculate degree days to trigger the onset.

`dd_onset_threshold`
: Degree days that are required to trigger the onset of infestation. Additionally, the maximum temperature must exceed `tfly`.

`tfly`
: Minimum temperature that beetles need to fly.

`dd_development_base`
: Base temperature to calculate degree days for development.

`dd_total_dev_lower`, `dd_total_dev_upper`
: Lower/upper limit of degree days that are required for a generation to fully develop

`dev_start`, `dev_end`
: Share in total development when the egg development starts and the juvenile beetle's development ends respectively. Usable if the development below/above these thresholds should account for mating, oviposition etc.

`dev_mortal_min`, `dev_mortal_max`
: The beetles are considered to be in white stages (egg, larva, pupa) if their development exceeds `dev_mortal_min` and subceeds `dev_mortal_max`. During these stages, the beetles could die due to a mortality event. `NULL` means that no lower/upper threshold is defined.

`model_end_date`
: Date when the model ends (no further development will be modeled).

`daylength_dia`, `tdia_min`
: When the daylength falls below `daylength_dia` and the average daily temperature falls below `tdia_min`, diapause will be initiated. The default value for the critical daylength was set to 19.3 hours according to Schroeder and Dalin (2017) who examined the photoperiodic diapause induction in Sweden. If the model is used for other regions, this value should be adjusted.

## Details

In barrks, model() is used to customize a model. The following code illustrates which parameters are available for the Jönsson model and specifies their default values.

```
model("joensson",

      # ==== onset ====

      dd_onset_start_date = '01-01',
      dd_onset_base = 5,
      dd_onset_threshold = 120,

      # ==== onset + development ====

      tfly = 20,

      # ==== development ====

      dd_development_base = 5,
      dd_total_dev_lower = 625,
      dd_total_dev_upper = 750,
      dev_start = 0,
      dev_end = 1,
      dev_mortal_min = 0,
      dev_mortal_max = 1,

      model_end_date = '12-31',

      # ==== diapause ====

      daylength_dia = 19.3,
      tdia_min = 15
)
```

## References

Jönsson AM, Harding S, Krokene P, Lange H, Åke Lindelöw A, Økland B, Ravn HP, Schroeder LM (2011). "Modelling the potential impact of global warming on *Ips typographus* voltinism and reproductive diapause." *Climatic Change*, **109**, 695-718. doi:10.1007/s1058401100384.

Schroeder M, Dalin P (2017). "Differences in photoperiod-induced diapause plasticity among different populations of the bark beetle *Ips typographus* and its predator *Thanasimus formicarius*." *Agricultural and Forest Entomology*, **19**(2), 146-153. doi:10.1111/afe.12189.

## See Also

model(), phenology(), model.joensson.apply

Other model customizations: model.bso.customize, model.chapy.customize, model.lange.customize, model.phenips.customize, model.phenips_clim.customize, model.rity.customize

---

model.lange.apply *Use the Lange model*

---

## Description

This page describes the usage of the the Lange model with phenology(). The model specific inputs are listed and its basic functionality is explained. The model was published by Lange et al. (2008) for *Ips typographus*.

## Arguments

tmin, tmean, tmax
: Daily minimum/mean/maximum temperatures in °C.

.submodels, .onset, .diapause, .mortality, ...
: See phenology() for a detailled description of the function.

## Details

In barrks, phenology() is used to apply a model. The following code illustrates which inputs are required to apply the Lange model and which additional parameters are available.

```
phenology("lange", ..., tmin, tmean, tmax)

# calculate submodels separately
phenology("lange", ..., .submodels = 'onset', tmean, tmax)
phenology("lange", ..., .submodels = 'mortality', tmin)
phenology("lange", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmean, tmax)
```

## Value

The function returns a phenology. Look here to find out how it can be analysed.

## Functioning

In the following, the basic functioning of the Lange model is explained.

- **Onset**: The onset of swarming is triggerd when the degree days of the maximum temperature reach a specific threshold and the maximum temperature exceeds the minimum flight temperature according to Annila (1969).
- **Development**: The development is calculated using stage-specific temperature sums and thresholds (Wermelinger and Seifert 1998). A new generation starts its development when the last generation finished its development and the maximum temperature exceeds the minimum flight temperature.

- **Diapause**: The Lange model does not have a diapause submodel implemented.
- **Mortality**: White stages (egg to pupa) die when the minimum temperature falls below a specific threshold.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

## References

Annila E (1969). "Influence of temperature upon the development and voltinism of Ips typographus L. (Coleoptera, Scolytidae)." *Annales Zoologici Fennici*, **6**(2), 161–208. http://www.jstor.org/stable/23731366.

Lange H, Økland B, Krokene P (2008). "To be or twice to be? The life cycle development of the spruce bark beetle under climate change." In *Unifying Themes in Complex Systems: Proceedings of the Sixth International Conference on Complex Systems*, 251–258. Springer. doi:10.1007/9783540850816_32.

Wermelinger B, Seifert M (1998). "Analysis of the temperature dependent development of the spruce bark beetle *Ips typographus* (L) (Col., Scolytidae)." *Journal of Applied Entomology*, **122**(1-5), 185-191. doi:10.1111/j.14390418.1998.tb01482.x.

## See Also

`model()`, `phenology()`, `model.lange.customize`

Other phenology applications: `model.bso.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.phenips.apply`, `model.phenips_clim.apply`, `model.rity.apply`

---

`model.lange.customize` *Customize the Lange model*

---

## Description

This page describes the parameters that can be used to customize Lange. The model was developed by Lange et al. (2008). Look here to find out how to apply the model.

## Arguments

| | |
|---|---|
| `dd_onset_start_date` | The date, when the degree days start to sum up ('MM-DD'). |
| `dd_onset_base` | Base temperature to calculate degree days to trigger the onset. |
| `dd_onset_threshold` | Degree days that are required to trigger the onset of infestation. Additionally, the maximum temperature must exceed `tfly`. |
| `tfly` | Minimum temperature that beetles need to fly. |

`dd_base_stages` Base temperatures to calculate degree days for the different stages in the following order: egg, larva, pupa, juvenile adult.

`dd_threshold_stages` Thermal thresholds for the different stages in the following order: egg, larva, pupa, juvenile adult.

`model_end_date` Date when the model ends (no further development will be modeled).

`first_lethal_date` Date before which no mortality will be modeled.

`tlethal` Temperature threshold below which white stages (egg, larva, pupa) will die.

## Details

In `barrks`, `model()` is used to customize a model. The following code illustrates which parameters are available for the Lange model and specifies their default values.

```
model("lange",

      # ==== onset ==== #

      dd_onset_start_date = '01-01',
      dd_onset_base = 5,
      dd_onset_threshold = 110,

      # ==== onset + development ====

      tfly = 19.5,

      # ==== development ==== #

      dd_base_stages = c(10.6, 8.2, 9.9, 3.2),
      dd_threshold_stages = c(51.8, 204.4, 57.7, 238.5),

      model_end_date = '12-31',

      # ==== mortality ==== #

      first_lethal_date = '09-01',
      tlethal = 0
)
```

## References

Lange H, Økland B, Krokene P (2008). "To be or twice to be? The life cycle development of the spruce bark beetle under climate change." In *Unifying Themes in Complex Systems: Proceedings of the Sixth International Conference on Complex Systems*, 251–258. Springer. doi:10.1007/9783-540850816_32.

## See Also

model(), phenology(), model.lange.apply

Other model customizations: model.bso.customize, model.chapy.customize, model.joensson.customize, model.phenips.customize, model.phenips_clim.customize, model.rity.customize

---

`model.phenips.apply` *Use PHENIPS*

---

## Description

This page describes the usage of PHENIPS with phenology(). The model specific inputs are listed and its basic functionality is explained. PHENIPS was published by Baier et al. (2007) and parametrized at the Kalkalpen National Park in Austria for *Ips typographus*.

## Arguments

| | |
|---|---|
| ... | See phenology() for a detailled description of the function. |
| tmean, tmax | Daily mean/maximum temperatures in °C. |
| rad | Daily radiation in W * h / m^2. |
| daylength | Length of the day in hours. Can be created with create_daylength_rst() or create_daylength_df(). |
| exposure | Specifies the sun exposure. Can be 'sunny' (default) or 'shaded'. |
| sister_broods | Set FALSE if sister broods should not be calculated. |

## Details

In barrks, phenology() is used to apply a model. The following code illustrates which inputs are required to apply PHENIPS and which additional parameters are available.

```
phenology("phenips", ..., tmean, tmax, rad, daylength,
          exposure = 'sunny', sister_broods = TRUE)

# calculate submodels separately
phenology("phenips", ..., .submodels = 'onset', tmax)
phenology("phenips", ..., .submodels = 'diapause', daylength)
phenology("phenips", ..., .submodels = 'mortality', tmax)
phenology("phenips", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmean, tmax, rad,
          exposure = 'sunny', sister_broods = TRUE)
```

## Value

The function returns a phenology. Look here to find out how it can be analysed.

### Functioning

In the following, the basic functioning of PHENIPS is explained.

- **Onset**: The onset is triggered when the degree days of the maximum temperature reach a specific threshold and the maximum temperature exceeds the minimum flight temperature.
- **Development**: The beetles develop according to a slightly modified version of the optimum curve described by Wermelinger and Seifert (1998) depending on the bark temperature. The bark temperature is modeled based on mean and maximum temperature, global radiation and sun exposure. A new generation will emerge when the last generation is fully developed and the maximum temperature exceeds the minimum flight temperature.
- **Diapause**: The diapause is initiated when the daylength falls below a threshold.
- **Mortality**: White stages (egg to pupa) die on a fixed date.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

### References

Baier P, Pennerstorfer J, Schopf A (2007). "PHENIPS—A comprehensive phenology model of *Ips typographus* (L.)(Col., Scolytinae) as a tool for hazard rating of bark beetle infestation." *Forest Ecology and Management*, **249**(3), 171–186. doi:10.1016/j.foreco.2007.05.020.

Wermelinger B, Seifert M (1998). "Analysis of the temperature dependent development of the spruce bark beetle *Ips typographus* (L) (Col., Scolytidae)." *Journal of Applied Entomology*, **122**(1-5), 185-191. doi:10.1111/j.14390418.1998.tb01482.x.

### See Also

`model()`, `phenology()`, `model.phenips.customize`

Other phenology applications: `model.bso.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips_clim.apply`, `model.rity.apply`

---

## `model.phenips.customize` *Customize PHENIPS*

---

### Description

This page describes the parameters that can be used to customize PHENIPS. The model was developed by Baier et al. (2007). Look here to find out how to apply the model.

## Arguments

`dd_onset_start_date`
: The date, when the degree days start to sum up ('MM-DD').

`dd_onset_base`
: Base temperature to calculate degree days to trigger the onset.

`dd_onset_threshold`
: Degree days that are required to trigger the onset of infestation. Additionally, the maximum temperature must exceed `tfly`.

`tfly`
: Minimum temperature that beetles need to fly.

`dd_development_base`
: Base temperature to calculate degree days for calculating the beetles development.

`dd_total_dev`
: Degree days that are required for a generation to fully develop

`dev_start`, `dev_end`
: Share in total development when the egg development starts and the juvenile beetle's development ends respectively. Usable if the development below/above these thresholds should account for mating, oviposition etc.

`dev_sister_brood`
: Share in the total development when a sister brood will be established.

`dev_mortal_min`, `dev_mortal_max`
: The beetles are considered to be in white stages (egg, larva, pupa) if their development exceeds `dev_mortal_min` and subceeds `dev_mortal_max`. During these stages, the beetles could die due to a mortality event. `NULL` means that no lower/upper threshold is defined.

`topt`
: Temperature for optimal development.

`tlow`, `tup`
: Temperature below/above which no development happens.

`func_btmean`, `func_btmax`, `func_btdiff`
: Functions to calculate the effective bark temperature (see Baier et al. 2007, equations A.3 to A5). Each parameter will be passed as SpatRaster:
  - `tmean`: mean air temperature
  - `tmax`: maximum air temperature
  - `rad`: radiation
  - `btmax`: maximum bark temperature

`model_end_date`
: Date when the model ends (no further development will be modeled).

`daylength_dia`
: When the daylength falls below this threshold, diapause will be initiated.

`mortality_date`
: Date when all white stages (egg, larva, pupa) die.

## Details

In `barrks`, `model()` is used to customize a model. The following code illustrates which parameters are available for PHENIPS and specifies their default values.

```
model("phenips",

      # ==== onset ====
```

```
  dd_onset_start_date = '04-01',
  dd_onset_base = 8.3,
  dd_onset_threshold = 140,

  # ==== onset + development ====

  tfly = 16.5,

  # ==== development ====

  dd_development_base = 8.3,
  dd_total_dev = 557,
  dev_start = 0,
  dev_end = 1,
  dev_sister_brood = 0.5,
  dev_mortal_min = NULL,
  dev_mortal_max = 0.6,

  topt = 30.4,
  tlow = 8.3,
  tup = 38.9,

  func_btmean = function(tmean, rad) {
    -0.173 + 0.0008518 * rad + 1.054 * tmean
  },
  func_btmax = function(tmax, rad) {
    1.656 + 0.002955 * rad + 0.534 * tmax + 0.01884 * tmax ^ 2
  },
  func_btdiff = function(btmax) {
    (-310.667 + 9.603 * btmax) / 24
  },

  model_end_date = '10-31',

  # ==== diapause ====

  daylength_dia = 14.5,

  # ==== mortality ====

  mortality_date = '10-31'
)
```

## References

Baier P, Pennerstorfer J, Schopf A (2007). "PHENIPS—A comprehensive phenology model of *Ips typographus* (L.)(Col., Scolytinae) as a tool for hazard rating of bark beetle infestation." *Forest Ecology and Management*, **249**(3), 171–186. doi:10.1016/j.foreco.2007.05.020.

## See Also

model(), phenology(), model.phenips.apply

Other model customizations: model.bso.customize, model.chapy.customize, model.joensson.customize, model.lange.customize, model.phenips_clim.customize, model.rity.customize

---

`model.phenips_clim.apply` *Use PHENIPS-Clim*

---

## Description

This page describes the usage of PHENIPS-Clim with phenology(). The model specific inputs are listed and its basic functionality is explained. PHENIPS-Clim is not published yet. This manual will be updated when a publication is available. It was parametrized for *Ips typographus* in southern Germany.

## Arguments

| | |
|---|---|
| ... | See phenology() for a detailled description of the function. See phenology() for details. |
| tmin, tmean, tmax | Daily minimum/mean/maximum temperatures in °C. tmin is optional. If available it will be used to calculate the temperature amplitude. If not, (tmax - tmean) * 2 will be used as amplitude. |
| rad | Daily radiation in W * h / m^2. |
| daylength | Length of the day in hours. Can be created with create_daylength_rst() or create_daylength_rst(). |
| sister_broods | Set FALSE to disable the calculation of sister broods. |
| scenario | Choose a scenario to use a suitable combination of parameters for specific situations. The scenario defines a default value for each value that can be overwritten by specifying a value for the respective parameter. The following scenarios are available:<br>• mean: list(exposure = 'sunny', onset_mode = 0.5, diapause_mode = 'photoperiodic', oviposition_mode = 0.5)<br>• max: list(exposure = 'sunny', onset_mode = 0.1, diapause_mode = 'thermal', oviposition_mode = 0.1) |
| exposure | Specifies the sun exposure. Can be 'sunny'(default) or 'shaded'. |
| onset_mode | Share of beetles that are already infesting trees necessary to trigger the onset. Must be 0.1, 0.5 or 0.9 if not customized. |
| oviposition_mode | Share of beetles that should have finished oviposition to trigger the beginning of the development. Must be 0.1, 0.5 or 0.9 if not customized. |
| diapause_mode | Determines how the diapause is initiated. Can be one of the following options: |

- 'photoperiodic': The diapause is initiated when the daylength falls below a specific threshold.
- 'thermal': The diapause is initiated by a logistic model that depends on the daylength and the maximum temperature.

Share of beetles that already stopped reproducing necessary to trigger the diapause. Must be thermal or 'photoperiodic' if not customized. If 'photoperiodic' is chosen, the diapause is controlled by a daylength threshold (see parameter daylength_dia here).

## Details

In barrks, phenology() is used to apply a model. The following code illustrates which inputs are required to apply PHENIPS-Clim and which additional parameters are available.

```
phenology("phenips-clim", ..., tmin, tmean, tmax, rad, daylength,
          sister_broods = TRUE, scenario = 'max', exposure = NULL,
          onset_mode = NULL, oviposition_mode = NULL, diapause_mode = NULL)

# calculate submodels separately
phenology("phenips-clim", ..., .submodels = 'onset', tmax, scenario = 'max', onset_mode = NULL)
phenology("phenips-clim", ..., .submodels = 'diapause', tmax, daylength, scenario = 'max', diapause_mod
phenology("phenips-clim", ..., .submodels = 'mortality', tmin)
phenology("phenips-clim", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmin, tmean, tmax, rad, sister_broods = TRUE,
          scenario = 'max', exposure = NULL, oviposition_mode = NULL)
```

## Functioning

In the following, the basic functioning of PHENIPS-Clim is explained.

- **Onset**: A base onset is triggered by a logistic model that relates to the maximum temperature and the respective degree days. Beginning from the base onset, a specific level of degree days (depending on the share of individuals that should be accounted for) and maximum air temperature must be reached to trigger the actual onset.
- **Development**: While the bark temperature and the emergence of new generations are determined according to PHENIPS, the calculation of the beetles' development rates is refined. Rather than implying a constant development within a day, temperature fluctuations are incorporated by taking the daily temperature amplitude into account. Additionally, the first part of development represents the pre-oviposition period and will not appear in the resulting output.
- **Diapause**: The diapause can be initiated due to the photoperiod according to PHENIPS or by a logistic model that depends on the daylength and the maximum temperature and accounts for beetles that reproduce even on shorter days if the temperatures are favorable. In the second case, PHENIPS-Clim detects a reproductive arrest, due to adverse abiotic parameters, and not an actual diapause as the output can be adjusted, if conditions improve and allow for further reproduction later in the season.
- **Mortality**: White stages (egg to pupa) die when the minimum temperature falls below a specific threshold.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

## See Also

`model()`, `phenology()`, `model.phenips_clim.customize`

Other phenology applications: `model.bso.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips.apply`, `model.rity.apply`

---

`model.phenips_clim.customize`
*Customize PHENIPS-Clim*

---

## Description

In `barrks`, `model()` is used to customize a model. Here, the parameters are described that can be used to customize PHENIPS-Clim. The model is currently unpublished. This manual will be updated as soon as a publication is available. Look here to find out how to apply the model.

## Arguments

`dd_onset_start_date`
: The date, when the degree days start to sum up ('MM-DD').

`dd_onset_base`
: Base temperature to calculate degree days to trigger the onset.

`onset_func`
: Function with the SpatRasters `tmax` (maximum temperature) and `dd_tmax` (degree days of maximum temperature) as parameters. The function should return `TRUE` when the base onset is triggered. See `onset_add_dd` for the actual onset of infestation.

`dd_onset_alt_start_date`, `dd_onset_alt_base`, `onset_alt_func`
: Alternative way to calculate the diapause (see `dd_onset_start_date`, `dd_onset_base` and `onset_func`). The first of both onset variants will be used. Set `onset_alt_func = NULL` to disable the alternative onset calculation.

`onset_add_dd`
: Vector of options to calculate the actual onset of infestation. The vector should be named after the share of beetles that already started breeding when the onset is triggered (choose an option via `phenology(..., onset_mode = [option])` when applying the model). The values specify the degree days that are required starting at the first positive return value of `onset_func`.

`tfly`
: Minimum temperature that beetles need to fly.

`dd_total_dev`
: Degree days that are required for a generation to fully develop

`dev_oviposition`
: Named numeric vector of shares in the total development when the oviposition is finished. The vector should be named after the share of beetles that should be taken into account (choose an option via `phenology(..., oviposition_mode = [option])` when applying the model).

`dev_end` Share in total development when the juvenile beetle's development ends. Usable if the development above this threshold should account for mating, oviposition etc.

`dev_sister_brood`
Share in the total development, when a sister brood will be established.

`dev_mortal_min`, `dev_mortal_max`
The beetles are considered to be in white stages (egg, larva, pupa) if their development exceeds `dev_mortal_min` and subceeds `dev_mortal_max`. During these stages, the beetles could die due to a mortality event. `NULL` means that no lower/upper threshold is defined.

`topt` Temperature for optimal development.

`func_btmean`, `func_btmax`, `func_btdiff`
Functions to calculate the bark temperatures (see Baier et al. 2007, equations A.3 to A.5). Each parameter will be passed as SpatRaster:

- `tmean`: mean air temperature
- `tmax`: maximum air temperature
- `rad`: radiation
- `btmax`: maximum bark temperature

`dev_rates` Data frame that specifies the development rates per day depending on the mean temperature and the temperature amplitude. Column names are the mean temperatures and row names the temperature amplitudes both with one decimal place. base onset (see `onset_func`) to trigger the actual onset.

`model_end_date` Date when the model ends (no further development will be modeled).

`first_diapause_date`
Date before which an initiation of the diapause is impossible ('MM-DD').

`diapause_thermal_func`
Function to calculate the initiation of the diapause if the model was applied using `phenology(..., diapause_mode = 'thermal')`. The diapause will be initiated the last time when the function returns `TRUE`.

`daylength_dia` When the daylength falls below this threshold, diapause will be initiated if the model was applied using `phenology(..., diapause_mode = 'photoperiodic')`.

`tlethal` Temperature threshold below which white stages will die.

## Details

In `barrks`, `model()` is used to customize a model. The following code illustrates which parameters are available for PHENIPS-Clim and specifies their default values.

```
model("phenips-clim",

      # ==== onset ====

      dd_onset_start_date = '03-01',
      dd_onset_base = 12,
      onset_func = \(tmax, dd_tmax) {
```

```
  0.564071 * tmax + 0.006434 * dd_tmax - 12.37046 > 0
},

dd_onset_alt_start_date = '04-01',
dd_onset_alt_base = 8.3,
onset_alt_func = \(tmax, dd_tmax) dd_tmax >= 140,

onset_add_dd = c('0.1' = 0, '0.5' = 90, '0.9' = 190),

# ==== development ====

tfly = 16.5,

dd_total_dev = 557,

dev_oviposition = c('0.1' = 0.1,
                    '0.5' = 0.15,
                    '0.9' = 0.26),
dev_end = 1,
dev_sister_brood = 0.3,

dev_mortal_min = NULL,
dev_mortal_max = 0.6,

topt = 30.4,

func_btmean = function(tmean, rad) {
  -0.173 + 0.0008518 * rad + 1.054 * tmean
},
func_btmax = function(tmax, rad) {
  1.656 + 0.002955 * rad + 0.534 * tmax + 0.01884 * tmax ^ 2
},
func_btdiff = function(tmax) {
  (-310.667 + 9.603 * tmax) / 24
},

# dev_rates too large to show here, type `params('phenips-clim')$dev_rates`
# to get the dev_rates that are used by default
# dev_rates = matrix(...),

model_end_date = '12-31',

# ==== diapause ====

first_diapause_date = '08-12',
diapause_thermal_func = function(daylength, tmax) {
  0.8619156 * daylength + 0.5081128 * tmax - 23.63691 > 0
},
```

```
        daylength_dia = 14.5,

        # ==== mortality ====

        tlethal = -5
)
```

## References

Baier P, Pennerstorfer J, Schopf A (2007). "PHENIPS—A comprehensive phenology model of *Ips typographus* (L.)(Col., Scolytinae) as a tool for hazard rating of bark beetle infestation." *Forest Ecology and Management*, **249**(3), 171–186. doi:10.1016/j.foreco.2007.05.020.

## See Also

`model()`, `phenology()`, `model.phenips_clim.apply`

Other model customizations: `model.bso.customize`, `model.chapy.customize`, `model.joensson.customize`, `model.lange.customize`, `model.phenips.customize`, `model.rity.customize`

---

`model.rity.apply` *Use RITY*

---

## Description

This page describes the usage of RITY with `phenology()`. The model specific inputs are listed and its basic functionality is explained. RITY (also called RITY-2) was published by Ogris et al. (2019) and parametrized for *Ips typographus* in Slovenia.

## Arguments

| | |
|---|---|
| `tmin`, `tmean`, `tmax` | Daily minimum/mean/maximum temperatures in °C. For the `development` submodel, the parameter that is obligatory depends on `mode`. |
| `daylength` | Length of the day in hours. Can be created with `create_daylength_rst()` or `create_daylength_rst()`. |
| `mode` | Specifies which temperature should be used to calculate the development. Can be `min`, `mean` or `max`. |
| `.submodels`, `.onset`, `.diapause`, `.mortality`, `...` | See `phenology()` for a detailled description of the function. |

## Details

In `barrks`, `phenology()` is used to apply a model. The following code illustrates which inputs are required to apply RITY and which additional parameters are available.

```
phenology("rity", ..., tmin = NULL, tmean = NULL, tmax, daylength, mode = 'max')

# calculate submodels separately
phenology("rity", ..., .submodels = 'onset', tmax)
phenology("rity", ..., .submodels = 'diapause', daylength)
phenology("rity", ..., .submodels = 'mortality', tmax)
phenology("rity", ..., .submodels = 'development',
          .onset, .diapause = NULL, .mortality = NULL,
          tmin = NULL, tmean = NULL, tmax = NULL, mode = 'max')
```

## Value

The function returns a phenology. Look here to find out how it can be analysed.

## Functioning

In the following, the basic functioning of RITY is explained.

- **Onset**: See PHENIPS.
- **Development**: Based on PHENIPS with a few modifications:
  - The optimum curve is calculated according to Wermelinger and Seifert (1998) without simplification.
  - The minimum, mean or maximum bark temperature can be used to calculate the development. These temperatures depend only on the respective air temperatures.
- **Diapause**: See PHENIPS.
- **Mortality**: See PHENIPS.

Look here to find out how the model parameters affect the actual calculations and which values are used by default.

## References

Ogris N, Ferlan M, Hauptman T, Pavlin R, Kavčič A, Jurc M, De Groot M (2019). "RITY–A phenology model of Ips typographus as a tool for optimization of its monitoring." *Ecological Modelling*, **410**, 108775. doi:10.1016/j.ecolmodel.2019.108775.

Wermelinger B, Seifert M (1998). "Analysis of the temperature dependent development of the spruce bark beetle *Ips typographus* (L) (Col., Scolytidae)." *Journal of Applied Entomology*, **122**(1-5), 185-191. doi:10.1111/j.14390418.1998.tb01482.x.

## See Also

`model()`, `phenology()`, `model.rity.customize`

Other phenology applications: `model.bso.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips.apply`, `model.phenips_clim.apply`

`model.rity.customize` *Customize RITY*

## Description

This page describes the parameters that can be used to customize RITY (also called RITY-2). The model was developed by Ogris et al. (2019). Look here to find out how to apply the model.

## Arguments

`dd_onset_start_date`
: The date, when the degree days start to sum up ('MM-DD').

`dd_onset_threshold`
: Degree days that are required to trigger the onset of infestation. Additionally, the maximum temperature must exceed `tfly`.

`tfly`
: Minimum temperature that beetles need to fly.

`dd_onset_base`
: Base temperature to calculate degree days for development.

`dd_total_dev`
: Degree days that are required for a generation to fully develop

`dev_start`, `dev_end`
: Share in total development when the egg development starts and the juvenile beetle's development ends respectively. Usable if the development below/above these thresholds should account for mating, oviposition etc.

`dev_sister_brood`
: Share in the total development, when a sister brood will be established.

`dev_mortal_min`, `dev_mortal_max`
: The beetles are considered to be in white stages (egg, larva, pupa) if their development exceeds `dev_mortal_min` and subceeds `dev_mortal_max`. During these stages, the beetles could die due to a mortality event. `NULL` means that no lower/upper threshold is defined.

`func_ftmean`, `func_ftmax`, `func_atdiff`
: Functions to caclulate the air temperature in forest stands (see Ogris et al. 2019, equations 1 - 3). Each parameter will be passed as SpatRaster:
  - `tmin`: min air temperature
  - `tmean`: mean air temperature
  - `tmax`: maximum air temperature

`func_btmean`, `func_btmax`, `func_btdiff`
: Functions to caclulate the bark temperature (see Ogris et al. 2019, equations 4 - 6). Each parameter will be passed as SpatRaster:
  - `ftmin`: min air temperature in forest stands
  - `ftmean`: mean air temperature in forest stands
  - `ftmax`: maximum air temperature in forest stands

`dt_low`, `dt_up`, `topt`, `tmax`, `alpha`, `beta`, `gamma`
: Parameters to calculate the effective bark temperature (see Ogris et al. 2019, equations 7 - 9).

| | |
|---|---|
| `model_end_date` | Date when the model ends (no further development will be modeled). |
| `daylength_dia` | When the daylength falls below this threshold, diapause will be initiated. |
| `mortality_date` | Date when all white stages (egg, larva, pupa) die. |

## Details

In barrks, `model()` is used to customize a model. The following code illustrates which parameters are available for RITY and specifies their default values.

```
model("rity",

      # ==== onset ====

      dd_onset_start_date = '03-07',
      dd_onset_base = 8.3,
      dd_onset_threshold = 155.6,

      # ==== onset + development ====

      tfly = 14.5,

      # ==== development ====

      dd_development_base = 8.3,
      dd_total_dev = 557,
      dev_start = 0,
      dev_end = 1,
      dev_sister_brood = 0.5,
      dev_mortal_min = NULL,
      dev_mortal_max = 0.6,

      func_ftmin = function(tmin) { 1.44 + 0.82 * tmin },
      func_ftmean = function(tmean) { 0.50 + 0.81 * tmean },
      func_ftmax = function(tmax) { 1.03 + 0.86 * tmax },

      func_btmin = function(ftmin) { 0.56 + 0.99 * ftmin },
      func_btmean = function(ftmean) { -0.48 + 1.03 * ftmean },
      func_btmax = function(ftmax) { 0.03 + 0.99 * ftmax },

      dt_low = 8.3,
      dt_up = 38.9,
      topt = 30.4,
      tmax = 40.9958913,
      alpha = 0.02876507,
      beta = 3.5922336,
      gamma = 1.24657367,

      model_end_date = '10-31',
```

```
        # ==== diapause ====

        daylength_dia = 14.5,

        # ==== mortality ====

        mortality_date = '10-31'
)
```

## References

Ogris N, Ferlan M, Hauptman T, Pavlin R, Kavčič A, Jurc M, De Groot M (2019). "RITY–A phenology model of Ips typographus as a tool for optimization of its monitoring." *Ecological Modelling*, **410**, 108775. doi:10.1016/j.ecolmodel.2019.108775.

## See Also

`model()`, `phenology()`, `model.rity.apply`

Other model customizations: `model.bso.customize`, `model.chapy.customize`, `model.joensson.customize`, `model.lange.customize`, `model.phenips.customize`, `model.phenips_clim.customize`

---

# model_combine *Combine different (sub-)models*

---

## Description

Combine different (sub-)models.

## Usage

```
model_combine(...)
```

## Arguments

| | |
|---|---|
| `...` | Phenology models, model names or lists with the keys `model` and `submodels`. In the last case, only the submodels specified are used (one of `'onset'`, `'diapause'`, `'mortality'` or `'development'`) of the respective model. If multiple models are supplied for the same submodel, the last one overwrites all others. |

## Value

A phenology model. Can be passed to `phenology()`.

## See Also

- model(), phenology()
- Customize (sub-)models: model.bso.customize, model.phenips.customize, model.rity.customize, model.chapy.customize, model.joensson.customize, model.lange.customize, model.phenips_clim.customi
- Use (sub-)models: model.bso.apply, model.phenips.apply, model.rity.apply, model.chapy.apply, model.joensson.apply, model.lange.apply, model.phenips_clim.apply

## Examples

```
# combine PHENIPS with the diapause submodel of PHENIPS-Clim
m <- model_combine('phenips',
                   list(model = 'phenips-clim', submodels = 'diapause'))

# calculate phenology
p <- phenology(m, barrks_data(), .quiet = TRUE)

# plot calculated generations
gens <- get_generations_rst(p)
terra::plot(gens)
```

---

# params *Get model parameters*

---

## Description

Get the parameters of a model.

## Usage

```
params(m, ...)
```

## Arguments

| | |
|---|---|
| m | Name of the model or the return value of another model()-call. |
| ... | List of parameters to customize the model. |

## Value

A list.

## Examples

```
# print the first parameters of `phenips-clim`
head(params('phenips-clim'))
```

phenology *Central function to calculate a phenology*

## Description

Calculate a phenology (or its subparts) with a specific model.

## Usage

```
phenology(
  .model,
  .data = NULL,
  .dates = NULL,
  .win = NULL,
  .ext = "tif",
  .onset = NULL,
  .diapause = NULL,
  .mortality = NULL,
  .submodels = c("onset", "diapause", "mortality", "development"),
  .setup_only = FALSE,
  .stations = NULL,
  .storage = NULL,
  .quiet = FALSE,
  ...
)

bso_phenology(
  .model = "bso",
  .data = NULL,
  .dates = NULL,
  .win = NULL,
  .ext = "tif",
  .onset = NULL,
  .diapause = NULL,
  .mortality = NULL,
  .submodels = c("onset", "diapause", "mortality", "development"),
  .setup_only = FALSE,
  .stations = NULL,
  .storage = NULL,
  .quiet = FALSE,
  ...
)
```

## Arguments

| | |
|---|---|
| .model | A phenology model or a model name (see model(), model_combine()). |
| .data | Data that will be passed to the model. It can be one of the following: |

- Character string: The raster data will be loaded from the path specified. The files have to be named like the respective model inputs.
- Named list: Each element contains the input data according to its name.
- Data frame (station data): Should have the columns `date` and `station` (name of the station). Additional columns have to be named like the respective model inputs.
- Additionally, data can be passed through the ... argument.

Look at the model application manuals to find out which inputs are required by a specific model: `model.bso.apply`, `model.phenips.apply`, `model.rity.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips_clim.apply`.

`.dates` Vector of dates that the data should be restricted to.

`.win` SpatExtent to set a window (area of interest) if `.data` is a path to load the raster data from.

`.ext` Extension of the files that should be used if `.data` is a path to load the raster data from.

`.onset`, `.diapause`, `.mortality`
Pass custom or precalculated phenological events to the model. See `create_events` to find out how to create events manually. Alternatively, the return value of `get_onset_rst()`, `get_diapause_rst()` or `get_mortality_rst()` could be used (with `as_doy = FALSE`) to extract the respective phenological event from another phenology. In that case, that phenology must match the temporal and spatial extent of the other inputs.

`.submodels` Character vector. Specifies which submodels should be calculated. Can be a subset of `c('onset', 'diapause', 'mortality', 'development')`.

`.setup_only` If `TRUE` only the inputs will be preprocessed without calculating any submodels. The preprocessed data can be used as input for other `phenology()` calls and can be accessed via `get_input_data()`.

`.stations` Assign stations to the phenology. See `stations_create()` for details.

`.storage` If set, the path specified here will be used to save the (intermediate) results. If `phenology()` is called successively with a growing amount of data, the calculations will continue where they stopped. This can save calculation time especially for large raster inputs. Note that this will only work of raster inputs and if `terra::sources()` is not empty. Otherwise the results of the calculations will be saved but successive calculations are not possible. If no input data is passed, the phenology will be loaded from the storage.

`.quiet` If `TRUE`, messages are suppressed.

`...` Parameters that will be passed to the model. Must be named according to the model inputs. See `.data` for alternative ways to pass data to the model.

## Value

A phenology as a list. Look here to find out how a phenology can be analysed. It is not recommended to access the list elements directly.

### Functions

- `bso_phenology()`: As BSO works a bit different than the other models, a seperate phenology function is implemented for this model. Note that while the onset and the development submodels are needed to be taken from BSO, the diapause and the mortality submodels are compatible with other models.

  The function returns a BSO phenology as a list. Look here to find out how a BSO phenology can be analysed. It is not recommended to access the list elements directly. To be able to use the functions that are available for phenology objects returned by `phenology()`, call `bso_translate_phenology()`.

### See Also

`model.bso.apply`, `model.phenips.apply`, `model.rity.apply`, `model.chapy.apply`, `model.joensson.apply`, `model.lange.apply`, `model.phenips_clim.apply`

### Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data())

# plot calculated generations
gens <- get_generations_rst(p)
terra::plot(gens)
```

---

## plot_development_diagram

*Plot a development diagram*

---

### Description

A development diagram illustrates the beetles' development of all appearing generations within a year.

### Usage

```
plot_development_diagram(
  .phenos,
  .station = prop_stations(.phenos[[1]])[1],
  .generations = NULL,
  .colors = barrks_colors("diagram_lines"),
  .fill = barrks_colors("diagram_fill"),
  .labels = barrks_labels("diagram"),
  .legend_col = TRUE,
  .legend_lty = TRUE,
  .group = TRUE,
  .minmax_only = FALSE,
```

```
  .fun_bg = NULL,
  .lty = "solid",
  .lwd = 2,
  .date_split = NULL,
  .date_stop = NULL,
  .lty2 = "dotted",
  .lwd2 = 2,
  .fill2 = NA,
  ...
)
```

## Arguments

| | |
|---|---|
| `.phenos` | List of (named) phenology objects or a single phenology that will be plotted (see `phenology()`). |
| `.station` | Pass a character vector to choose a station assigned to pheno by its name, or pass a different station. See `stations_create()` for details. |
| `.generations` | Generations that will be shown. |
| `.colors`, `.fill`, `.labels` | Character vectors. Change the line colors, fill or labels of the generations starting from the first generation followed consecutively by elements for the other generations (including sister broods). |
| `.legend_col`, `.legend_lty` | Manipulate the appearance of the legends for colors and line types. Pass `TRUE`/`FALSE` to enable/disable the respective legend. For the customization of the respective legend, a list of parameters for graphics::legend can be passed. |
| `.group` | Select the phenology objects that will be used to draw the filling. It can be a character vector of the phenology names, an integer vector of the phenology numbers, or `TRUE` if all phenology objects should be used. |
| `.minmax_only` | If `TRUE`, only the minimum and the maximum development line will be plotted. |
| `.fun_bg` | Function to draw a background. |
| `.lty`, `.lwd` | Use specific line types and line widths. Vectors of the same length as `.phenos` will assign the values to the respective phenology. |
| `.date_split`, `.lty2`, `.lwd2`, `.fill2` | When `.date_split` is reached, the appearance of the plot will change according to the respective values. |
| `.date_stop` | If specified, no data will be plotted after the respective date. |
| `...` | Parameters passed to `base::plot()`. |

## Value

None

## See Also

stations

## Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data('stations'), .quiet = TRUE)

# plot development diagram of the station 'Mannheim'
plot_development_diagram(p, 'Mannheim', .lwd = 4, .legend_lty = FALSE)
```

---

`properties` *Get phenology properties*

---

## Description

To examine a phenology, there are different functions to query its properties.

## Usage

```
## get the year the phenology was calculated for
prop_year(pheno)

## get all dates that are covered by the phenology
prop_dates(pheno)

## get the first date that is covered by the phenology
prop_first_date(pheno)

## get the last date that is covered by the phenology
prop_last_date(pheno)

## get all hatched generations as numeric vector
prop_hatched_generations(pheno)

## get all hatched filial generations as numeric vector
prop_filial_generations(pheno)

## get all hatched sister broods as numeric vector
prop_sister_broods(pheno)

## get the stations assigned to the phenology
prop_stations(pheno)
```

## Arguments

| | |
|---|---|
| pheno | A phenology (see `phenology()`) |

## Value

The requested property.

## Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)

# print all generations that were hatched
prop_hatched_generations(p)
```

---

`save_phenology` *Save/load a phenology*

---

## Description

Saves/loads a phenology to/from a path.

## Usage

```
save_phenology(
  pheno,
  .storage,
  .submodels = c("onset", "diapause", "mortality", "development"),
  .overwrite = FALSE,
  .ext = ".tif",
  .quiet = FALSE
)

load_phenology(
  .storage,
  .submodels = c("onset", "diapause", "mortality", "development"),
  .ext = ".tif",
  .quiet = FALSE
)
```

## Arguments

| | |
|---|---|
| `pheno` | A phenology, calculated with `phenology()`. |
| `.storage` | Path to save/load the phenology. |
| `.submodels` | Which submodels should be saved/loaded. |
| `.overwrite` | Should an existing storage be overwritten? |
| `.ext` | Extension for raster files. |
| `.quiet` | If `TRUE`, messages are suppressed. |

## Value

- `save_phenology()`: None
- `load_phenology()`: A phenology as a list. Look here to find out how a phenology can be analysed. It is not recommended to access the list elements directly.

### Functions

- `save_phenology()`: Saves a phenology to a path.
- `load_phenology()`: Loads a phenology from a path.

### Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)

# choose path to save the phenology
path <- file.path(tempdir(), 'pheno')

# save phenology
save_phenology(p, path, .overwrite = TRUE, .quiet = TRUE)


###

# load phenology from path
p2 <- load_phenology(path, .quiet = TRUE)

# plot generations
gens <- get_generations_rst(p2)
terra::plot(gens)
```

---

## `stations_create` *Work with stations*

---

### Description

In `barrks`, stations are references to specific raster cells. Thus, they can be used to extract point-related data from a phenology. Look here to find out which station-based functions are available to analyse a phenology.

### Usage

```
stations_create(station_names, cells)

stations_assign(pheno, stations)

stations_names(stations)

stations_cells(stations)
```

## Arguments

| | |
|---|---|
| station_names | Character vector that specifies the names of the stations. |
| cells | Numbers of the cells that should be represented by the stations. |
| pheno | A phenology (see phenology()) |
| stations | Stations created with stations_create() or obtained by prop_stations(). |

## Value

- stations_create(): A named numeric vector.
- stations_assign(): A phenology object (see phenology()).
- stations_names(): A character vector.
- stations_cells(): A numeric vector.

## Functions

- stations_create(): Create stations.
- stations_assign(): Assign stations to a phenology. Returns the phenology that was passed with respective stations assigned.
- stations_names(): Get the names of stations.
- stations_cells(): Get the raster cells of stations.

## Examples

```
# calculate phenology
p <- phenology('phenips-clim', barrks_data(), .quiet = TRUE)

# create stations and assign them to the phenology object
stations <- stations_create(c('station a', 'station b'),
                           c(234, 345))
p <- stations_assign(p, stations)

# plot the development of 'station b'
plot_development_diagram(p, 'station b', .lwd = 4, .legend_lty = FALSE)
```

# Index


∗ **model customizations**
  model.bso.customize, 27
  model.chapy.customize, 31
  model.joensson.customize, 35
  model.lange.customize, 38
  model.phenips.customize, 41
  model.phenips_clim.customize, 46
  model.rity.customize, 51
∗ **phenology applications**
  model.bso.apply, 26
  model.chapy.apply, 29
  model.joensson.apply, 33
  model.lange.apply, 37
  model.phenips.apply, 40
  model.phenips_clim.apply, 44
  model.rity.apply, 49

analyse.phenology, 3, *5*
analyse.phenology.bso, *4*, 4

barrks_colors, 5
barrks_colors(), *7*
barrks_data, 6
barrks_labels, 7
barrks_labels(), *5*
base::plot(), *11*, *58*
bso_get_flight *(*bso_get_flight_rst*)*, 7
bso_get_flight_df *(*bso_get_flight_rst*)*, 7
bso_get_flight_df(), *5*
bso_get_flight_rst, 7
bso_get_flight_rst(), *4*
bso_get_individuals *(*bso_get_individuals_rst*)*, 8
bso_get_individuals_df *(*bso_get_individuals_rst*)*, 8
bso_get_individuals_df(), *5*, *11*
bso_get_individuals_rst, 8
bso_get_individuals_rst(), *4*
bso_phenology *(*phenology*)*, 55
bso_phenology(), *4*, *8–12*, *23*, *27*
bso_plot_flight_diagram, 9
bso_plot_flight_diagram(), *5*
bso_plot_stage_diagram, 10
bso_plot_stage_diagram(), *5*
bso_translate_phenology, 12
bso_translate_phenology(), *57*

categorize_generations_rst, 13
create_daylength_df, 14
create_daylength_df(), *15*, *40*
create_daylength_rst, 15
create_daylength_rst(), *14*, *30*, *33*, *40*, *44*, *49*
create_diapause *(*create_onset*)*, 16
create_events, *56*
create_events *(*create_onset*)*, 16
create_mortality *(*create_onset*)*, 16
create_onset, 16
create_suntimes_df, 17
create_suntimes_df(), *19*, *26*
create_suntimes_rsts, 18
create_suntimes_rsts(), *18*, *26*

get_development *(*get_development_rst*)*, 19
get_development_df *(*get_development_rst*)*, 19
get_development_df(), *4*
get_development_rst, 19
get_development_rst(), *3*
get_diapause *(*get_onset_rst*)*, 23
get_diapause_df *(*get_onset_rst*)*, 23
get_diapause_df(), *4*
get_diapause_rst *(*get_onset_rst*)*, 23
get_diapause_rst(), *3*, *56*
get_events *(*get_onset_rst*)*, 23
get_generations *(*get_generations_rst*)*, 20

get_generations_df (get_generations_rst), 20
get_generations_df(), 4
get_generations_rst, 20
get_generations_rst(), 3
get_hibernating_generations_df (get_generations_rst), 20
get_hibernating_generations_df(), 4
get_hibernating_generations_rst (get_generations_rst), 20
get_hibernating_generations_rst(), 3
get_input_data, 22
get_input_data(), 56
get_mortality (get_onset_rst), 23
get_mortality_df (get_onset_rst), 23
get_mortality_df(), 4
get_mortality_rst (get_onset_rst), 23
get_mortality_rst(), 3, 56
get_onset (get_onset_rst), 23
get_onset_df (get_onset_rst), 23
get_onset_df(), 4
get_onset_rst, 23
get_onset_rst(), 4, 56
graphics::legend, 11, 58

here, 12, 26, 27, 31, 34, 35, 37, 38, 40, 41, 45, 46, 50, 51, 56, 57, 60, 61

list_models, 24
load_phenology (save_phenology), 60

model, 25
model(), 27–30, 32, 33, 35, 36, 38–42, 44, 46, 47, 49, 50, 52–55
model.bso.apply, 26, 29, 30, 35, 38, 41, 46, 50, 54, 56, 57
model.bso.customize, 25, 27, 27, 33, 37, 40, 44, 49, 53, 54
model.chapy.apply, 27, 29, 33, 35, 38, 41, 46, 50, 54, 56, 57
model.chapy.customize, 25, 29, 30, 31, 37, 40, 44, 49, 53, 54
model.joensson.apply, 27, 30, 33, 36, 38, 41, 46, 50, 54, 56, 57
model.joensson.customize, 25, 29, 33, 35, 35, 40, 44, 49, 53, 54
model.lange.apply, 27, 30, 35, 37, 40, 41, 46, 50, 54, 56, 57
model.lange.customize, 25, 29, 33, 37, 38, 38, 44, 49, 53, 54
model.phenips.apply, 27, 30, 35, 38, 40, 44, 46, 50, 54, 56, 57
model.phenips.customize, 25, 29, 33, 37, 40, 41, 41, 49, 53, 54
model.phenips_clim.apply, 27, 30, 35, 38, 41, 44, 49, 50, 54, 56, 57
model.phenips_clim.customize, 25, 29, 33, 37, 40, 44, 46, 46, 53, 54
model.rity.apply, 27, 30, 35, 38, 41, 46, 49, 53, 54, 56, 57
model.rity.customize, 25, 29, 33, 37, 40, 44, 49, 50, 51, 54
model_combine, 53
model_combine(), 55

parametrization, 30
params, 54
PHENIPS, 45, 50
phenology, 55
phenology(), 3, 12, 16, 20–23, 25, 26, 29, 30, 33–38, 40, 41, 44–46, 49, 50, 53, 54, 56–60, 62
plot_development_diagram, 57
plot_development_diagram(), 4
prop_dates (properties), 59
prop_dates(), 3, 4
prop_filial_generations (properties), 59
prop_filial_generations(), 3, 4
prop_first_date (properties), 59
prop_first_date(), 3, 4
prop_hatched_generations (properties), 59
prop_hatched_generations(), 3, 4
prop_last_date (properties), 59
prop_last_date(), 3, 4
prop_sister_broods (properties), 59
prop_sister_broods(), 3, 4
prop_stations (properties), 59
prop_stations(), 3, 4, 62
prop_year (properties), 59
prop_year(), 3, 4
properties, 59

RITY, 30

save_load_phenology (save_phenology), 60
save_phenology, 60

stations, *58*
stations *(*stations_create*)*, 61
stations_assign *(*stations_create*)*, 61
stations_cells *(*stations_create*)*, 61
stations_create, 61
stations_create(), *8–11*, *16*, *20, 21*, *23*, *56*, *58*
stations_names *(*stations_create*)*, 61

terra::time(), *15*, *19*