Package 'DEPONS2R'

September 8, 2025

Type Package

Title Read, Plot and Analyse Output from the DEPONS Model

Version 1.2.8

Description Methods for analyzing population dynamics and movement tracks simulated using the DEPONS model <https://www.depons.eu> (v.3.0), for manipulating input raster files, shipping routes and for analyzing sound propagated from ships.

License GPL-3

Encoding UTF-8

LazyData true

LazyDataCompression xz

Depends R (>= 3.5.0)

Imports raster, methods, sp, sf, terra, utils, grDevices, xml2, jsonlite, adehabitatLT, adehabitatHR

RoxygenNote 7.3.2

Suggests testthat (>= 3.0.0)

Config/testthat/edition 3

NeedsCompilation no

Author Jacob Nabe-Nielsen [aut, cre], Caitlin K. Frankish [aut], Axelle Cordier [aut], Florian G. Weller [aut]

Maintainer Jacob Nabe-Nielsen <jnn@ecos.au.dk>

Repository CRAN

Date/Publication 2025-09-08 15:40:13 UTC

Contents

ais.to.DeponsShips 3
aisdata 4
argosmetrics 5

as.data.frame,DeponsDyn-method . . . 5
as.data.frame,DeponsTrack-method . . . 6
as.DeponsRaster . . . 7
as.raster . . . 7
bathymetry . . . 8
bbox . . . 9
calib_01 . . . 9
calib_02 . . . 10
check.DeponsShips . . . 11
coastline . . . 13
crs . . . 13
DEPONS2R . . . 14
DeponsBlockdyn-class . . . 14
DeponsDyn-class . . . 15
DeponsRaster-class . . . 16
DeponsShips-class . . . 17
DeponsTrack-class . . . 17
dyn . . . 18
get.latest.sim . . . 18
get.simtime . . . 19
interpolate.ais.data . . . 20
interpolate.maps . . . 21
landscape<- . . . 23
make.blocksraster . . . 23
make.clip.poly . . . 25
make.construction.traffic . . . 26
make.DeponsDyn . . . 29
make.stationary.ships . . . 30
make.windfarms . . . 33
plot,DeponsBlockdyn,missing-method . . . 34
plot,DeponsDyn,missing-method . . . 35
plot,DeponsRaster,ANY-method . . . 36
plot,DeponsShips,missing-method . . . 37
plot,DeponsTrack,missing-method . . . 38
plot_calib02 . . . 39
porpoisebdyn . . . 40
porpoisedyn . . . 40
porpoisetrack . . . 41
read.DeponsBlockdyn . . . 41
read.DeponsDyn . . . 42
read.DeponsDynBatch . . . 44
read.DeponsParam . . . 45
read.DeponsRaster . . . 46
read.DeponsShips . . . 47
read.DeponsTrack . . . 47
read.DeponsTrackBatch . . . 49
routes . . . 50
set.ship.type . . . 51

shipdata 52
ships 52
startday 53
Summary-methods 53
tick.to.time 54
time.to.tick 55
title<- 56
write,DeponsShips-method 57

Index **59**

`ais.to.DeponsShips` *Convert ship tracks to DeponsShips object*

Description

Convert Automatic Identification System (AIS) data for ships to ship track objects. This is done by cropping one or more ship tracks to the extent of a landscape and converting the data to a `DeponsShips-class` object. If the AIS data does not include ship positions recorded in half-hour steps, the tracks are interpolated to make objects suitable for use in DEPONS.

Usage

```
ais.to.DeponsShips(data, landsc, title = "NA", ...)
```

Arguments

`data`	data.frame with ship positions and the times at which the positions were recorded. Must contain the columns 'id', 'time' (of the form " type, character), 'length' (ship length, meters), 'x', and 'y' (position, meters/UTM).
`landsc`	A `DeponsRaster` object corresponding to the landscape that the ships move in. It is assumed that the spatial projection of the ship positions corresponds to that of the DeponsRaster object
`title`	Title of the output object
`...`	Optional parameters, including 'startday' and 'endday' (" from 'data'. If startday = endday the output object will contain up to 49 positions from the selected date for each vessel track.

Value

Returns a `DeponsShips` object containing one or more ships assigned to each of the routes in the object. All ships on a particular route move at the same speed along the route. The routes are defined by x and y coordinates based on the same coordinate reference system as the landscape they are located in. The speed that ships use after reaching a particular position (a particular 'virtual buoy') is calculated from the distance to the following position, and the time it takes reaching that position. If speed is included in the input AIS data, this is NOT used. The routes include one position per half-hour time step, corresponding to the default time step used in the DEPONS model.

If input data does not include one position per half hour, new positions are generated using linear interpolation. If the input data contains many positions in a particular half-hour interval, only the positions closest to the half-hour interval are used. The routes contain information about the number of half-hour intervals were ships 'pause' at a particular location, e.g. in a port. These are calculated based on the input AIS data.

See Also

aisdata for an example of data that can be used as input to ais.to.DeponsShips. The function builds on interpolate.ais.data, which interpolates tracks to ensure that there is a position every 30 minutes. Use check.DeponsShips for testing if speeds are realistic. See write.DeponsShips for conversion of DeponsShips objects to json-files to be used in DEPONS. Use routes, ships, and title for inspection/modification of the ship tracks.

Examples

```
data(aisdata)
plot(aisdata$x, aisdata$y, type="n", asp=1)
ids <- sort(unique(aisdata$id))
my.colors <- heat.colors(length(ids))
for (i in 1:length(ids)) {
  id <- ids[i]
  points(aisdata$x[aisdata$id==id], aisdata$y[aisdata$id==id],
     cex=0.6, col=my.colors[i])
}
data(bathymetry)
plot(bathymetry, add=TRUE)
depons.ais <- ais.to.DeponsShips(aisdata, bathymetry)
the.routes <- routes(depons.ais)
for (i in 1:length(ids)) {
points(the.routes[[i]]$x, the.routes[[i]]$y,
        cex=0.6, pch=16, col=my.colors[i])
}
depons.ais <- ais.to.DeponsShips(aisdata, bathymetry,
   startday="2015-12-20", endday="2015-12-20")
routes(depons.ais)
aisdata2 <- aisdata
aisdata2$time <- format(as.POSIXct(aisdata$time)+300)
depons.ais2 <- ais.to.DeponsShips(aisdata2, bathymetry,
                                  startday="2015-12-20", endday="2015-12-21")
routes(depons.ais2)
```

---

aisdata *Position for three ships in the inner Danish waters*

Description

Automatic identification system (AIS) data for three ships in Kattegat and the Western Baltic from 20 Dec 2015. The data set includes the variables id (the Maritime Mobile Service Identity number),

time, speed (in knots), type, length (in metres), x and y (which provide the coordinates of the ship at a given time. The coordinates use the UTM zone 32 projection (CRS = "+proj=utm +zone=32 +units=m +no_defs +datum=WGS84").

Format

data.frame

`argosmetrics` *Fine and Large-scale metrics of Argos data*

Description

This dataset is a list of two dataframes containing fine and large-scale movements metrics of satellite tracked animals (Argos). For fine-scale movements, metrics include home range (HR, km2), mean net squared displacement (NSD, km2) and residence time (Rt, days) on 30-days tracks where animals used only fine-scale movements. For large-scale movements, metrics include HR (km2), max NSD (km2), a sinuosity index (Benhamou, 2004) and cumulative distance moved (km) on 100-days tracks that are independent from movement types.

Format

list

`as.data.frame,DeponsDyn-method`
Convert DeponsDyn object to data frame

Description

Function for converting DEPONS population dynamics object to a data frame.

Usage

```
## S4 method for signature 'DeponsDyn'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

`x`	DeponsDyn object
`row.names`	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
`optional`	Logical (not used)
`...`	additional arguments to be passed to or from methods.

Value

data.frame object

Examples

```
data(porpoisedyn)
class(porpoisedyn)
the.dyn <- as.data.frame(porpoisedyn)
```

---

as.data.frame,DeponsTrack-method

Convert DeponsTrack to data frame

Description

Function for converting DEPONS movement track file to a data frame.

Usage

```
## S4 method for signature 'DeponsTrack'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	DeponsTrack object
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	Logical (not used)
...	additional arguments to be passed to or from methods.

Value

data.frame object

Examples

```
data(porpoisetrack)
class(porpoisetrack)
the.track <- as.data.frame(porpoisetrack)
```

as.DeponsRaster *Convert a RasterLayer into a DeponsRaster*

Description

Converts a RasterLayer into a DeponsRaster.

Usage

```
## S4 method for signature 'RasterLayer'
as.DeponsRaster(x)
```

Arguments

x	A RasterLayer

Details

Maintains CRS (if defined) and value of NA cells. Currently DEPONS requires a fixed cell size of 400 x 400 m, and cell size is set to this value.

Value

A DeponsRaster

See Also

as.raster for converting a DeponsRaster into a RasterLayer

Examples

```
data(bathymetry)
bathymetry_RasterLayer <- as.raster(bathymetry)
bathymetry <- as.DeponsRaster(bathymetry_RasterLayer)
```

as.raster *Convert a DeponsRaster into a RasterLayer*

Description

Converts a DeponsRaster into a RasterLayer for easier manipulation with common R tools.

Usage

```
## S4 method for signature 'DeponsRaster'
as.raster(x)
```

Arguments

x	A DeponsRaster

Details

Maintains cell size / resolution, CRS (if defined) and value of NA cells.

Value

A RasterLayer

See Also

as.DeponsRaster for converting a RasterLayer into a DeponsRaster

Examples

```
data(bathymetry)
y <- as.raster(bathymetry)
```

---

bathymetry *Bathymetry of the Kattegat area*

Description

The standard bathymetry file for Kattegat which is used in DEPONS simulations. It is based on a raster file with 1000 rows and 600 columns where each grid cell corresponds to 400 m x 400 m. Cells on land are assigned a missing data value of -9999.

The Kattegat landscapes use the UTM zone 32 projection, (EPSG:32632) as in the study by Nabe-Nielsen et al (2014). The corresponding proj4string is "+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs" (see https://epsg.io/32632).

Format

DeponsRaster

References

Nabe-Nielsen, J., Sibly, R. M., Tougaard, J., Teilmann, J., & Sveegaard, S. (2014). Effects of noise and by-catch on a Danish harbour porpoise population. Ecological Modelling, 272, 242–251. doi:10.1016/j.ecolmodel.2013.09.025

See Also

DeponsRaster-class

bbox *Get bbox from Depons* object*

Description

Retrieves spatial bounding box from object. If a Depons* object is a DeponsTrack object containing multiple track, the box bounds all tracks.

Usage

```
## S4 method for signature 'DeponsRaster'
bbox(obj)

## S4 method for signature 'DeponsTrack'
bbox(obj)
```

Arguments

obj	DeponsRaster or DeponsTrack object

Value

Returns a `matrix` defining the northern, southern, eastern and western boundary of a DeponsRaster object or of one or more DeponsTrack objects.

See Also

`make.clip.poly`

calib_01 *Plot distribution of turning angles, step lengths and speed of tracked simulated porpoises.*

Description

Plot distribution of turning angles, step lengths and speed of tracked simulated porpoises.

Usage

```
calib_01(depons_track)
```

Arguments

depons_track	Object of class ‘DeponsTrack‘ produced by either read.DeponsTrack or read.DeponsTrackBatch

Value

Plot histograms of turning angles, step length and speed. It also stores those metrics of the animal in a dataframe.

`calib_02` *Calculate calibration metrics of previously filtered tracks (fine-scale or large-scale)*

Description

Calculate calibration metrics of previously filtered tracks (fine-scale or large-scale)

Usage

```
calib_02(track_cleaned, option)
```

Arguments

`track_cleaned`	A dataframe of the filtered track (either fine scale of large scale).
`option`	A character string, either ‘"fine"‘ or ‘"large"‘, indicating which type of movement (fine-scale or large-scale) metrics to return.‘"fine"‘ returns home range, mean net squared displacement, and mean residence time. ‘"large"‘ returns home range, maximum net squared displacement, sinuosity and cumulative distance moved.

Value

A dataframe storing either fine scale metrics (home range (HR, km2), mean net squared displacement (NSD, km2) and mean residence time (RT, days)) or large scale metrics (HR, max NSD, sinuosity index and cumulative distance moved (km)).

Examples

```
## Not run:
# filtering fine-scale movements on porpoise tracks

data(porpoisetrack)
track <-as.data.frame(porpoisetrack)
track$year <- track$tick / 17280
track$yearRound <- floor(track$year) + 1
track$tickNew <- ave(track$tick, track$Id, track$yearRound, FUN = seq_along)
track$day <- track$tickNew / 48
track$dayRound <- floor(track$day) + 1
track$date <- as.POSIXct("2014-01-01 00:00:00", tz = "GMT") + (track$tick * 1800)
track_fine <- track[track$DispersalMode == 0, ]

noon_tracks <- track_fine[format(track_fine$date, "%H:%M:%S") == "12:00:00", ]
```

```
# identify 30 consecutive days with noon data
all_days <- sort(unique(as.Date(noon_tracks$date)))
consecutive_days <- NULL

for (i in 1:(length(all_days) - 29)) {
if (all(diff(all_days[i:(i + 29)]) == 1)) {
 consecutive_days <- all_days[i:(i + 29)]
 break
}
}
# filter data to only the 30 consecutive days
filtered_tracks <- noon_tracks[as.Date(noon_tracks$date) %in% consecutive_days, ]

calib_02(filtered_tracks, option = "fine")

## End(Not run)
```

`check.DeponsShips` *Check if ships move at unrealistic speeds or are outside the map boundary*

Description

Checks if calculated speeds in DeponsShips objects are unrealistic, which may result from inaccurate AIS positional records or from ships leaving the map area, then re-entering at a remote position. As ship speed in DEPONS directly influences the amount of noise generated, it is advisable to detect and remove such instances to avoid the creation of extreme noise sources. The function can also repair issues arising from ship positions that are a fraction of a meter outside the map boundary (causing loading errors on simulation start).

Usage

```
check.DeponsShips(
  x,
  threshold = 35,
  fix = F,
  replacements = NA,
  landscape = NULL
)
```

Arguments

`x`	DeponsShips object
`threshold`	The speed (knots) above which calculated values are considered unrealistic/excessive. Defaults to 35 knots.
`fix`	Logical. If FALSE (default), the function returns a data frame of ship tracks containing speeds that exceed the threshold; if TRUE, the function returns a DeponsShips object where these instances have been replaced.

`replacements` Named list, where names are ship types and values are replacement speeds (knots) for speeds above the threshold within those types. Only ship types named in the list are processed. If NA (default), reference speeds from Table 1 in MacGillivray & de Jong (2021) are used.

`landscape` DeponsRaster object. Optional; a map representative of the simulation map extent (usually the bathymetry map). If provided and fix = TRUE, ship positions on the boundary will be adjusted to avoid errors from fractional mis-positioning.

Details

The default replacement speeds (knots) for recognized ship types are as follows (class reference speeds from MacGillivray & de Jong, 2021, Table 1): Fishing, 6.4; Tug, 3.7; Naval, 11.1; Recreational, 10.6; Government/Research, 8; Cruise, 17.1; Passenger, 9.7; Bulker, 13.9; Containership, 18.0; Tanker, 12.4; Dredger, 9.5; Other, 7.4.

If a simulation fails during data loading with an error that indicates ship positions outside the simulation area, this may be caused by a mismatch in rounding between the map extent of the map used with `ais.to.DeponsShips`, and of generated ship position exactly on the boundary. If a map representative of the simulation area extent is provided (usually the bathymetry map), the function will also repair these positions by rounding them up/down to the floor/ceiling of the map extent (fractional meter adjustments).

Value

If fix = FALSE, a data frame with columns "route number", "name", "type", "length", and "speed", containing one entry for each ship where an excessive speed occurred. If fix = TRUE, a DeponsShip object where instances of excessive speed have been replaced, and (if a map has been provided) where ship positions on the boundary have been adjusted.

Reference

MacGillivray, A., & de Jong, C (2021). A reference spectrum model for estimating source levels of marine shipping based on Automated Identification System data. Journal of Marince Science and Engineering, 9(4), 369. doi:10.3390/jmse9040369

See Also

`ais.to.DeponsShips` for creation of DeponsShips objects (including calculated speeds) from AIS data

Examples

```
## Not run:
x <- shipdata
check.DeponsShips(x)

x@routes$route[[1]]$speed <- x@routes$route[[1]]$speed * 3
check.DeponsShips(x)
x <- check.DeponsShips(x, fix = T)
## End(Not run)
```

coastline *Coastline of Northern Europe*

Description

An object of class SpatialPolygonsDataFrame showing the coastline of the North Sea, Kattegat, and the Western Baltic. The map projection used is ETRS89 – EPSG:3035 projection as for the North Sea raster files used by DEPONS. The corresponding proj4string is "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000 +datum=WGS84 +units=m +no_defs".

Format

SpatialPolygonsDataFrame

crs *Get or set map projection in Depons* objects*

Description

Get or set the map projection (also known as coordinate reference system, crs) of DeponsRaster and DeponsTrack objects.

Usage

```
## S4 method for signature 'DeponsTrack'
crs(x)

## S4 method for signature 'DeponsShips'
crs(x)

## S4 method for signature 'DeponsRaster'
crs(x)

## S4 replacement method for signature 'DeponsTrack'
crs(x) <- value

## S4 replacement method for signature 'DeponsShips'
crs(x) <- value

## S4 replacement method for signature 'DeponsRaster'
crs(x) <- value
```

Arguments

x	Object of class class DeponsRaster, DeponsShips or DeponsTrack
value	(proj4string) identifying the map projection

DEPONS2R *Package for analyzing DEPONS simulation output*

Description

Methods for analyzing population dynamics and movement tracks simulated using the DEPONS model (v.3.0; https://www.depons.eu), for manipulating input raster files, shipping routes and for analyzing sound propagated from ships.

The classes used in DEPONS2R include:

- DeponsTrack movement tracks, read from "RandomPorpoise.XXX.csv" files
- DeponsDyn population dynamics data, from "Statistics.XXX.csv" files
- DeponsBlockdyn data from "PorpoisePerBlock.XXX.csv" files
- DeponsShips data from "ships.json" files or from AIS data
- DeponsRaster stores raster landscape files used in DEPONS

Here the DeponsDyn data include both changes in population size and energetics through time for the entire landscape, whereas DeponsBlockdyn data include variations in population size in different parts (or 'blocks') of the landscape. XXX is the date and time when the simulation was finished.

DeponsBlockdyn-class *DeponsBlockdyn-class*

Description

Stores objects containing population size for different parts of the landscape (i.e. different 'blocks')

Details

The dyn slot contains a data frame with the columns 'tick', which indicates the number of half-hourly time steps since the start of the simulation; a column 'block' indicating the region of the landscape where animals were counted, a 'count' column with the number of animals in that block and tick. The 'real.time' column shows the real-world equivalent to 'tick, i.e. the time that has passed since 'startday'.

Slots

title Character. Name of the object or simulation

landscape Character. Identifier for the landscape used in the DEPONS simulations. The landscapes 'DanTysk', 'Gemini', 'Kattegat', 'North Sea', 'Homogeneous', and 'User defined' are distributed with the DEPONS model.

simtime POSIXlt object with the date and time when the simulation was finished. This is read from the name of the imput file.

startday POSIXlt object with the first day of the simulation, i.e. the first day in the period that the simulations are intended to represent in the real world.

dyn Data frame with simulation output.

Note

DeponsBlockdyn-objects are usually read in from csv files produced during DEPONS simulations. These files are named 'PorpoisePerBlock.XXX.csv', where XXX indicates the date and time when the simulation was finished.

See Also

`plot.DeponsBlockdyn` and `read.DeponsBlockdyn`.

Examples

```
a.DeponsBlockdyn <- new("DeponsBlockdyn")
a.DeponsBlockdyn
```

DeponsDyn-class *DeponsDyn-class*

Description

Stores objects containing population dynamics output and energetic output simulated using the DEPONS model.

Details

The following columns are included in the simulation output data frame: 'tick', which indicates the number of half-hourly time steps since the start of the simulation; 'count', which indicates the population size at a given time; 'anim.e', showing the average amount of energy stored by simulated animals; 'lands.e', which shows the total amount of energy in the landscape, and 'real.time' which shows the time relative to 'startday'.

Slots

`title` Character. Name of the object or simulation

`landscape` Character. Identifier for the landscape used in the DEPONS simulations. The landscapes 'DanTysk', 'Gemini', 'Kattegat', 'North Sea', 'Homogeneous', and 'User defined' are distributed with the DEPONS model.

`simtime` `POSIXlt` object with the date and time when the simulation was finished. This is read from the name of the imput file.

`startday` POSIXlt object with the first day of the simulation, i.e. the first day in the period that the simulations are intended to represent in the real world.

`dyn` Data frame with simulation output.

Note

DeponsDyn-objects are usually read in from csv files produced during DEPONS simulations. These files are named 'Statistics.XXX.csv', where XXX indicates the date and time when the simulation was finished.

See Also

plot.DeponsDyn and read.DeponsDyn.

Examples

```
a.DeponsDyn <- new("DeponsDyn")
a.DeponsDyn
```

DeponsRaster-class *DeponsRaster-class*

Description

Stores objects containing raster landscapes used as input in DEPONS simulations.

Slots

`type` Character. Identifies the kind of data stored in the raster; should be 'food', 'patches', bathymetry', 'dtc', 'salinity', 'blocks' or 'NA'.

`landscape` Character Identifier for the landscape used in the DEPONS simulations. The landscapes 'DanTysk', 'Gemini', 'Kattegat', 'North Sea', 'Homogeneous', and 'User defined' are distributed with the DEPONS model.

`crs` Object of class "CRS", i.e. the coordinate reference system. This is provided as a proj4string text string.

`header` Data frame with data on number of columns and rows in the input raster, the coordinates of the lower left corner, the size of each grid cell and the integer value used to represent missing data.

`ext` Data frame with the extent of the landscape.

`data` The actual data values for each of the grid cells.

Note

DeponsRaster-objects are typically read in from ascii raster files that have been used for DEPONS simulations.

See Also

plot.DeponsRaster, read.DeponsRaster and make.blocksraster. bathymetry is an example of a DeponsRaster-object.

Examples

```
a.deponsraster <- new("DeponsRaster")
a.deponsraster
```

DeponsShips-class *DeponsShips-class*

Description

Objects containing ship routes and ships

Methods for manipulating, plotting and analyzing ship routes and ship agents used in DEPONS simulations.

Slots

`title` Name of the object (character)

`landscape` Name of the landscape that the ships occur in (character)

`crs` CRS object providing the coordinate reference system used; see `CRS` for details

`routes` `data.frame` geographic positions of the 'virtual buoys' that define one or more ship routes that ship agents follow, and the speed that the ships should use when following this route. They also provide information about how long ships should use speed zero when reaching a specific buoy ('i.e. 'pause', measured in minutes). Can be extracted using the `routes` function.

`ships` `data.frame` defining each of the ships occurring in DEPONS simulations, and the routes they occur on. The data frame includes the variables 'name', 'type', 'length', and 'route'. Info can be extracted using the `ships` function.

See Also

`plot.DeponsShips`, and `read.DeponsShips`

Examples

```
data(shipdata)
ships(shipdata)[1:10,]
routes(shipdata)
plot(shipdata, col=c("red", "purple", "blue"))
```

DeponsTrack-class *DeponsTrack-class*

Description

Stores objects containing animal movement tracks simulated using the DEPONS model

Classes for manipulating and plotting movement tracks generated with DEPONS.

Slots

`title` Name of the object (character)

`landscape` Name of the object (character)

`simtime` POSIXlt object with the date and time when the simulation was finished. This is read from the name of the imput file.

`crs` CRS object providing the coordinate reference system used; see `st_crs` for details

`tracks` Listwith one or more tracks, each stored as a `SpatialPointsDataFrame` object)

See Also

`plot.DeponsTrack` and `read.DeponsTrack`

---

dyn *Extract population dynamics from objects*

Description

Extract population dynamics from objects

Usage

```
## S4 method for signature 'DeponsDyn'
dyn(x)

## S4 method for signature 'DeponsBlockdyn'
dyn(x)
```

Arguments

`x`	Object of class `DeponsBlockdyn`.

---

get.latest.sim *Get name of newest file*

Description

Returns the name of the newest simulation output of a particular type within the specified directory. The date and time are extracted from the file name.

Usage

```
get.latest.sim(type = "dyn", dir)
```

Arguments

type	Type of simulation output to check; can be one of: "dyn" (for looking in "Statistics.XX.csv" files), "blockdyn" (for looking in "PorpoisePerBlock.XX.csv" files) "track" (for looking in "RandomPorpoise.XX.csv" files).
dir	Directory to look for simulation output in (character string)

Value

character string with the name of the most recent simulation output file.

See Also

read.DeponsBlockdyn for example.

---

get.simtime *Get simulation date*

Description

Returns the date and time when a specific simulation was finished, obtained from the date stored as part of the file name. The date format is system dependent, but the function attemts to extract the data assuming that either the English or the local language is used. (a POSIXlt object)

Usage

```
get.simtime(fname = NULL, tz = "UTC")
```

Arguments

fname	Character string with name of the file to extract the simulation date from, including the path
tz	Time zone

Value

Returns a POSIXlt object

See Also

get.latest.sim

`interpolate.ais.data` *Interpolate AIS data*

Description

Interpolates ship movement tracks obtained from Automatic Identification System (AIS) data to obtain exactly one position per 30 minutes. The first and last position in the original track are omitted unless minutes = 0 or 30 and seconds = 0.

Usage

```
interpolate.ais.data(aisdata)
```

Arguments

`aisdata`	Data frame including the columns 'id' (ship identifier), 'time' (text string readable by `as.POSIXct`), 'x' and 'y' (recorded ship position, unit: meters), and potentially additional columns

Value

Returns a data frame with the same columns as the input data. Tracks that are too short to interpolate are omitted (with a warning)

See Also

`read.DeponsShips` and `ais.to.DeponsShips`

Examples

```
data(aisdata)
ais.testdata <- aisdata[c(12,14,16) ,]
plot(ais.testdata[c("x", "y")], asp=1, col="green", pch=16, xlim=c(780000, 837000))
lines(ais.testdata[c("x", "y")])
# Add 600 sec to 'time' to mis-allign with intervcal needed
ais.testdata$time <- format(as.POSIXlt(ais.testdata$time, tz = "UTC")+600)
text(ais.testdata[c("x", "y")]-900, ais.testdata$time, adj=0, cex=0.5)
interpolated <- interpolate.ais.data(ais.testdata)
points(interpolated[,c("x", "y")], col="red")
text(interpolated[c("x", "y")]-900, interpolated$time, adj=0, cex=0.5)
legend("bottomright", bty="n", pch=c(16, 1), col=c("green", "red"),
    legend=c("original positions", "interpolated"))
```

`interpolate.maps` *Interpolate missing entries in a series of maps*

Description

Interpolates values for missing entries in a series of maps that is to be read by DEPONS. This is intended for map series that in their entirety are assumed to follow a cyclical pattern (e.g., monthly prey or temperature maps, where the complete series covers one year) and therefore uses an interpolation that fits a single-cycle sine curve that seamlessly connects the last to the first map in the series.

Usage

```
interpolate.maps(map.dir, map.files, flatten = T, supplement = NA)
```

Arguments

`map.dir`	Character vector. The directory containing the source files and to which the output will be written.
`map.files`	Character vector. Complete map series in chronological order, where existing maps are given as file names and missing maps are given as 'NA'. All file formats supported by `raster::raster` are recognized.
`flatten`	Logical. Default TRUE. If true, negative interpolated values are set to 0. This prevents the generation of interpolated data that make no sense in context (negative food or temperature values). Since the fitted sine curve is likely to create some negative predictions, it is suggested to always flatten the output.
`supplement`	Numerical. Default NA. Can be used to specify one missing map (position number in map.files) that is to be used to 'spike' the data for a second round of fitting and prediction (see Details).

Details

The complete series of maps, consisting of existing and missing maps, is arranged as a chronological stack. For each raster cell in the stack, the prediction function is fit to the existing values, and the missing values are predicted from the function. The fitted function is of the form 'α * sine(x) + β * cosine(x)', where α & β are estimated using a basic linear model. This is equivalent to 'Amplitude * sine(x + Phase)', but is more straightforward to estimate. The resulting curve satisfies the assumption that the data described in the map series should have a single series maximum and minimum, and should loop around seamlessly at end of series.

The function requires at least 3 existing maps in the series to interpolate from. Number of total entries and missing entries is arbitrary.

The function writes a map file for each map in the series, where existing maps are unchanged and formerly missing maps consist of interpolated values. Created rasters are written in .asc format and named "interpolated_x.asc", where x is the number in the series. The coordinate system is inherited. NA values are set to -9999.

If a map is specified in 'supplement', it is created by interpolation in the first round, then the original data are supplemented with this map, and the fitting and interpolation are carried out again for the complete series. This can be helpful if the existing data describe the shape of the full series poorly. For example, if only the first three month of a twelve-month series of maps are given, a sine curve fitted to these may have no existing data describing a reasonable annual maximum or minimum, and estimated values may therefore be unrealistic. In this case it may be useful to estimate a first fit with 'flatten = T' (default), spike the data with the month that is expected to contain the minimum (which has been limited to 0 now), and fit the curve again, providing the function with a more realistic existing minimum to shoot for and thus constraining the amplitude of the curve.

Value

No return value (called for side effects)

Examples

```
## Not run:
## generate incomplete series of map files
map1 <- raster::raster(matrix(nrow = 100, ncol = 100))
for (i in 10:1) {
 map1[1:(i*10),1:(i*10)] <- 10 + i
}
map3 <- map1 + 10
map7 <- map1 - 5
maps <- replicate(12, NA, simplify = F)
maps[c(1,3,7)] <- c(map1, map3, map7)
par(mfrow=c(3,4), mar=c(0,0,0,0))
for (i in 1:12) {
 if (is.na(maps[i])) plot(0, type='n', axes=FALSE, ann=FALSE)
 else raster::plot(maps[[i]], col=rev(rainbow(40, start=0, end=1)),
 breaks=c(0:40), legend = F, axes = F)
}
raster::writeRaster(map1, "map1.asc", format = "ascii", overwrite = T)
raster::writeRaster(map3, "map3.asc", format = "ascii", overwrite = T)
raster::writeRaster(map7, "map7.asc", format = "ascii", overwrite = T)

## interpolate missing maps
interpolate.maps(map.dir = getwd(),
                 map.files = c("map1.asc", NA, "map3.asc", NA, NA, NA,
                 "map7.asc", NA, NA, NA, NA, NA))

par(mfrow=c(3,4), mar=c(0,0,0,0))
for (i in 1:12) {
 the.map <- paste0("interpolated_", i, ".asc")
 raster::plot(raster::raster(the.map), col=rev(rainbow(40, start=0, end=1)),
 breaks=c(0:40), legend = F, axes = F)
 unlink(the.map)
}
unlink(c("map1.asc", "map3.asc", "map7.asc"))

## End(Not run)
```

landscape<- *Get or set the landscape name*

Description

Get or set the landscape name

Get or set the landscape name

Usage

```
## S4 replacement method for signature 'DeponsTrack'
landscape(x) <- value

## S4 method for signature 'DeponsTrack'
landscape(x)

## S4 replacement method for signature 'DeponsDyn'
landscape(x) <- value

## S4 method for signature 'DeponsDyn'
landscape(x)

## S4 replacement method for signature 'DeponsBlockdyn'
landscape(x) <- value

## S4 method for signature 'DeponsBlockdyn'
landscape(x)
```

Arguments

x	Object of class DeponsBlockdyn.
value	Name of the landscape (character)

make.blocksraster *Makes new file with blocks*

Description

Produces a DeponsRaster object of type='blocks' for use in DEPONS simulations. This allows animals to be counted within specific regions (blocks) of the landscape during the simulation. The new blocks can be specified as either matrices or SpatialPolygons objects. For matrices, the blocks are defined as the smallest rectangle that includes all the specified positions.

Usage

```
## S4 method for signature 'DeponsRaster'
make.blocksraster(
  template,
  blocks = NA,
  blockvals = NULL,
  NAvalue = -9999,
  plot = FALSE,
  fname = NULL,
  overwrite = FALSE
)
```

Arguments

template	DeponsRaster object used as template for new blocks file
blocks	list of areas to be used for new blocks. Each item in 'blocks' should be a matrix (with two columns, corresponding to x- and y-coordinates) or a SpatialPolygons object
blockvals	Vector of integer values defining the labels of the new blocks. The first value defines the background value, so the length of 'blockvals' should equal the number of blocks plus 1
NAvalue	Value used for missing data in the output object
plot	If TRUE, the raster block is plotted
fname	Name of the output raster file (character string ending with '.asc'). No file is written to disk if fname is not provided.
overwrite	Whether to replace existing file.

Value

RasterLayer object defining different subregions of the landscape where animals should be counted.

Note

The blocks file should not be modified when running DEPONS simulations using the 'Kattegat' landscape. In this landscape the simulated animals use the blocks file for navigation. Also note that blocks are added to the new blocks raster in the order they are file in the order in which they are listed in 'blocks', so the order mattes if the blocks overlap.

Examples

```
#Load file to use as template for new blocks file
data("bathymetry")

# Make list of blocks to create
new.blocks <- list()
x <- runif(8, 700000, 760000)
y <- runif(8, 6200000, 6300000)
new.blocks[[1]] <- cbind(x,y)
```

```
x <- c(600000, 635000, 670000, 635000)
y <- c(6150000, 6200000, 6150000, 6100000)
library(sp)
srl <- list(Polygon(cbind(x,y)))
Srl <- list(Polygons(srl, ID=as.vector("p")))
new.blocks[[2]] <- SpatialPolygons(Srl, proj4string=crs(bathymetry))

make.blocksraster(bathymetry, new.blocks, plot=TRUE)
points(new.blocks[[1]])
plot(new.blocks[[2]], add=TRUE)


the.dir <- tempdir()
make.blocksraster(bathymetry, new.blocks, fname=paste0(the.dir, "/test.asc"))
```

`make.clip.poly` *Make clipping polygon from bbox*

Description

Makes a polygon from a bounding box to use for clipping the coastline, or other SpatialPolygons objects

Usage

```
## S4 method for signature 'matrix'
make.clip.poly(bbox, crs)
```

Arguments

`bbox`	2x2 matrix
`crs`	CRS object defining the projection of the SpatialPolygons object to be clipped

Value

`SpatialPolygons` object

See Also

`bbox` for creation of bbox matrix from DeponsRaster

Examples

```
data(bathymetry)
bbox <- cbind("min"=c(549517, 6155000), "max"=c(636000, 6210000))
rownames(bbox) <- c("x", "y")
clip.poly <- make.clip.poly(bbox, crs(bathymetry))
```

`make.construction.traffic` *Generate artificial AIS data representing ship traffic during wind farm construction*

Description

Generates artificial Automatic Identification System (AIS) data to represent ship traffic connected with the construction of a wind farm, to supplement scenarios where no real records of such data are available. A list of ship characteristics may be supplied, or a built-in default set of ships may be used. Ship routes are constructed such that all ships start at a chosen harbour position, attend each piling event observing individual stay durations, and between pilings either return to the harbour or traverse as much of the route back to the harbour as time allows. Route data can afterwards be converted to a `DeponsShips` object (using `ais.to.DeponsShips`) which can be read by DEPONS.

Usage

```
make.construction.traffic(
  pilings,
  ships = NULL,
  x.harbour,
  y.harbour,
  startday = "2010-01-01",
  tz = "UTC"
)
```

Arguments

`pilings`	A data frame containing the positions and times of piling operations during the construction of a wind farm. May contain real data but must be in the format as produced by `make.windfarms`: columns 'id', 'x.coordinate' (num), 'y.coordinate' (num), 'impact' (num; optional - not required for this function), 'tick.start' (num), and 'tick.end' (num).
`ships`	A data frame with the characteristics of the ships that should be simulated. Must contain one entry for each ship and the columns 'id', length' (num; ship length, meters), 'speed' (num; knots), and 'daily.pause' (num; length of active, i.e. noisy, pause at each piling event, ticks). If no data are provided, a set of 13 ships based on data from piling operations at the Moray East wind farm in 2019 is used (see details).
`x.harbour`	Numeric. X coordinates (meters) of originating harbour for all ships
`y.harbour`	Numeric. Y coordinates (meters) of originating harbour for all ships
`startday`	Character. Intended start date of the simulation. If ticks provided in 'pilings' were converted from real dates using `time.to.tick`, this should be the same as the 'origin' used in that conversion. Defaults to "2010-01-01".
`tz`	Time zone. Defaults to "UTC

Details

All ships will attend each piling event. The route of each ship over the wind farm construction period is created in this manner: 1) The ship will try to be in position at a piling for the duration of its 'pause.length'; this is considered an active pause, i.e. the ship holds position by engine use and generates noise. The pause is timed such that the piling event's midpoint is in the middle of the pause. 2) The ship starts at the harbour location at the beginning of the data set, and will leave for the first piling in time to reach it at its given speed. 3) Between any two piling events, the ship will attempt to go back to the harbour if there is sufficient time to do so, and wait there until the next event (see ship ID_1 in example). If there is insufficient time to cover the whole distance back and forth, the ship will go back as far as possible along the route, and turn around in time to reach the next piling event (see ship ID_2 in example). If the time between pilings is shorter than the active pause duration, the ship will move straight between piling events, spending half of the remaining time pausing at each location. (When assigning active pauses later using `make.stationary.ships`, pausing at the harbour will not be considered an active pause, i.e. no noise will be generated.) 4) After the last piling event, the ship will return to the harbour and wait there until the end of the simulation.

If the number of ticks until the first piling event is too low for the slowest ship in the set to reach it from the harbour, an error is thrown, in which case the time of all piling events should be shifted backward. If the ticks were converted from real dates using `time.to.tick`, this can be done by choosing an earlier 'origin' in that conversion.

Ships can only move in a straight line between the harbour and any piling event. The harbour position should therefore be chosen so that such routes do not cut across islands, headlands etc. (land intersections will however not interfere with movement, and the user may consider ignoring minor irregularities for the sake of convenience).

Function `make.windfarms` generates hypothetical piling data that can be used with this function. If real piling data are used, dates should be converted to ticks using `time.to.tick`.

The type of all ships is set to "Other", since this category includes those vessel classes expected to be present at piling operations.

After the data have been generated, the user's next step will probably be conversion to a `DeponsShips` object with `ais.to.DeponsShips`, and parameterization of active pauses using `make.stationary.ships`. It is recommended to also use `check.DeponsShips` to verify ship speeds.

If no ship data are provided, a set of 13 ships based on data from piling operations at the Moray East wind farm in 2019 is used:

vessel class	length (m)	active pause duration (ticks)
CTV	14	0
CTV	15	1
CTV	19	8
CTV	23	1
CTV	24	1
CTV	25	4
CTV	25.75	1
CTV	25.75	2
CTV	27	1
Dive	20	1
OS	88.4	11

OS	89	41
OS	120	9

(CTV: crew transfer vessel, Dive: dive support ship, OS: offshore supply ship)

The type of these ships is set to "Other" and the speed to 7.4 knots (see `make.stationary.ships` for the source of these conventions).

Ships remaining at sea for the duration of the complete piling operations

One or more principal construction ships may be intended to remain within the piling area for the entire duration of piling operations without returning to harbour. This is achieved by setting the ship's 'daily.pause' duration in the 'ships' dataframe to a high number of ticks (greater than the longest gap between pilings, e.g. multiple days of 48 ticks each), which will cause the ship to progress directly from one piling to the next. This value will be capped at 48 ticks before the first and after the last piling event. Note that all this time spent pausing at sea will be parameterized as an active (noisy) pause when the generated ship data set is later processed with `make.stationary.ships`. This may be correct for crane ships or similar that actively hold position at sea, but inappropriate for jack-up vessels that take up a fixed position at the piling. In the latter case, the user should make note of the ship's identifier, and after carrying out the 'check' step of processing with 'make.stationary.ships', remove all of the ship's entries from the candidates data frame before carrying out the 'replace' step (see `make.stationary.ships` for details).

Value

A data frame of ship positions and the times at which the positions are set, with columns 'id', 'time' (of the form " 'x', and 'y' (num; position, meters). This data frame is suitable for conversion using `ais.to.DeponsShips`.

See Also

`make.windfarms` for creation of hypothetical wind farm piling data.

Examples

```
x.harbour <- 0
y.harbour <- 0
ships <- as.data.frame(rbind(c("ID_1", 20, 14, 2),
                             c("ID_2", 20, 8, 20)))
ships[,2:4] <- as.numeric(unlist(ships[,2:4]))
colnames(ships) <- c("id", "length", "speed", "pause.length")
pilings <- as.data.frame(rbind(c("Piling_1", 100000, 100000, 50, 54),
                               c("Piling_2", 102000, 100000, 80, 84),
                               c("Piling_3", 100000, 102000, 110, 114),
                               c("Piling_4", 102000, 102000, 140, 144)))
pilings[,2:5] <- as.numeric(unlist(pilings[,2:5]))
colnames(pilings) <- c("id", "x.coordinate", "y.coordinate", "tick.start", "tick.end")
construction.traffic <- make.construction.traffic(pilings = pilings, ships = ships,
                                      x.harbour = x.harbour, y.harbour = y.harbour)
```

`make.DeponsDyn` *Make DeponsDyn object from data stored in data frame*

Description

Function for reading converting a data frame containing DEPONS simulation output to a DeponsDyn object.

Usage

```
make.DeponsDyn(
  oname,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  startday = "2000-01-01",
  timestep = 30,
  tz = "UTC"
)
```

Arguments

`oname`	Name of the object (data frame) that contains number of animals for each time step during the simulation, along with their energy and the amount of food in the landscape.
`title`	Optional character string giving name of simulation
`landscape`	The landscape used in the simulation
`simtime`	Optional character string with the date and time when the simulation finished (format yyyy-mm-dd).
`startday`	The start of the period that the simulation represents, i.e. the real-world equivalent of 'tick 1' (character string of the form 'yyyy-mm-dd', or POSIXlt). Defaluts to 2000-01-01 (UTC time).
`timestep`	Time step used in the model, in minutes. Defaults to 30 minutes in DEPONS.
`tz`	Time zone.

Value

`DeponsDyn` object containing simulation output

See Also

See `DeponsDyn-class` for details on what is stored in the output object.

Examples

```
data(porpoisedyn)
the.data <- as.data.frame(porpoisedyn)
the.data <- the.data[, c(1:4)]
names(the.data) <- c("tick", "PorpoiseCount", "FoodEnergyLevel", "PorpoiseEnergyLevel")
porpoisedyn2 <- make.DeponsDyn(the.data, startday="2010-01-01")
porpoisedyn2
```

---

make.stationary.ships *Identify and parameterize stationary active ships in a DeponsShips object*

Description

Identifies ship positions in a DeponsShips object where the ship is stationary (pausing) but potentially still actively using its engine (bollard pushing or using dynamic positioning system), and if desired assigns a suitable non-zero speed to ensure noise generation at that time point. Candidates may be found either among all ships that are at a minimum distance from shore, or among those that are close to specific structures of interest, such as wind turbines.

Usage

```
make.stationary.ships(
  x,
  action = "check",
  candidates = NULL,
  distcrit = "shore",
  landscape = NULL,
  structure_locations = NULL,
  start_day = NA,
  start_times = NULL,
  verbose = F
)
```

Arguments

x	DeponsShips object
action	Character. If "check" (default), returns a data frame of pause positions that are candidates for stationary activity based on the selected criteria. If "replace" and a candidates data frame is provided, returns a DeponsShip object where the pauses identified in the data frame have been converted to stationary active status (i.e., a non-zero speed has been assigned)
candidates	A data frame of pause positions that are candidates for stationary activity. Required if 'action = "replace"'. Generated by using 'action = "check"'

`distcrit` Character. Main criterion for finding candidates for stationary activity. If "shore" (default), all ship positions in open water are eligible, subject to a number of secondary criteria (see Details). In this case, a DeponsRaster must be provided that allows determination of distance from land (see below). If any other value or NA, only ship positions close to specified structure locations (such as turbine piles) are eligible, and these locations must be provided via 'structure_locations' (see below). In this case, a start day for the ship records and individual start times for the structure locations may also be provided to allow simulation of an ongoing construction process (see below)

`landscape` A DeponsRaster where land areas are indicated as NA (e.g., the prey map for the simulation). Required if 'distcrit = "shore"' to determine distance of candidate positions from land

`structure_locations` A data frame with columns "id", "x" (numerical) and "y" (numerical), and one row for each structure that is to be used as a proximity criterion for finding candidates. Required if distcrit != 'shore'

`start_day` A character string or POSIX object of the form 'YYYY-MM-DD HH:MM:SS'. Defines the start time of x. Optional; can be provided together with start_times if distcrit != 'shore', to allow checking whether structures under construction are present at a given time point

`start_times` A data frame with columns "time" (character string or POSIX of format 'YYYY-MM-DD HH:MM:SS') and "id", and one row for each structure that is to be used as a proximity criterion for finding candidates. Defines time from which onward the structure is present. Optional; can be provided together with start_day if distcrit != 'shore', to allow checking whether structures under construction are present at a given time point

`verbose` Logical (default False). If True, writes a summary of each candidate to the console during "check" runs

Details

When a DeponsShips object is created using `ais.to.DeponsShips`, positions are interpolated at 30-minute intervals (ticks). If a ship's position does not change during sequential ticks, these ticks are combined into a pause of the appropriate duration, with a movement speed of 0. However, in some cases, an unmoving ship is actually using its engine to hold position, such as a crew transfer vessel performing a bollard push against a turbine pile, or an offshore supply vessel using a dynamic position system (DPS). Under these circumstances, the ship should emit noise to affect porpoise agents. This function attempts to identify and rewrite such pausing instances in an existing DeponsShips object. A pause is converted into an active stationary position by assigning a non-zero speed and thus noise emission. Note that assigning a speed does not translate into movement, as movement in the model is only derived from position changes, and speed is only used to drive noise calculation.

The intended functionality is to first run the function using 'action = "check"' to return a table of candidate instances. After this has been inspected and thinned as desired by the user, the function is run again using 'action = "replace"' while providing the table as 'candidates', which returns a DeponsShips object where the identified candidate pauses have been replaced with speed values.

No testing criteria (distcrit, landscape, stucture_locations, start_day, start_times) are required for a "replace" run, as the instances provided as 'candidates' are then modified without further checks.

Only ships with type "Other" or "Government/Research" (following the key in Table 1 in MacGillivray & de Jong 2021) are tested, as these categories contain the survey, construction and crew transfer ships that are the primary candidate types. Passenger, recreational, fishing and cargo vessels are assumed to not or rarely use DPS and are omitted. However, the "Other" category also contains vessels that hold position for extended periods without using DPS, such as jack-up rigs and platforms; also, ship type codes provided in AIS data are frequently unreliable. We therefore strongly suggest that the user should carefully scrutinize the candidates table produced in a "check" run, look up vessels by their MMSI code, and remove any false positives from the table before processing it in a "replace" run.

The inserted speed values are 7.4 knots for "Other" and 8 knots for "Government/Research", based on the class reference speeds in MacGillivray & de Jong (2021).

When 'distcrit = "shore"', pause instances are additionally tested against the following criteria: 1) not in a cell (400x400 m) directly adjacent to land, to exclude berthed ships; 2) not in a cell at the map boundary, as `ais.to.DeponsShips` will create inactive (pausing) placeholder positions at the point of entry if a ship enters the map with a delay after the object's start, or at the point of exit if it leaves before the end of the object's duration; 3) not in the first or last position of the ship's track (same reason).

When candidates are identified based on proximity to a list of structures, a maximum distance of 97.72 m is allowed, based on an estimate of mean AIS positioning error (Jankowski et al. 2021).

Value

If 'action = "check"' (default), returns a data frame with columns "route_number", "ship_name", "ship_type", "route_pos" (position number along route), and "pauses" (number of pauses at this position), with one row for each position that is a candidate for stationary activity based on the selected criteria. If "replace" and a candidates data frame is provided, returns a DeponsShip object where the pauses identified in the data frame have been converted to stationary active status (i.e., a non-zero speed has been assigned).

References

MacGillivray, A., & de Jong, C (2021). A reference spectrum model for estimating source levels of marine shipping based on Automated Identification System data. Journal of Marince Science and Engineering, 9(4), 369. doi:10.3390/jmse9040369"

Jankowski, D, Lamm A, & Hahn, A (2021). Determination of AIS position accuracy and evaluation of reconstruction methods for maritime observation data. IFAC-PapersOnLine, 54(16), 97-104. doi:10.1016/j.ifacol.2021.10.079

See Also

`ais.to.DeponsShips` for creation of DeponsShips objects (including calculated speeds) from AIS data

Examples

```
## Not run:
data(shipdata)
data(bathymetry)
candidates <- make.stationary.ships(shipdata,
                                    landscape = bathymetry,
                                    verbose = T)
shipdata.updated <- make.stationary.ships(shipdata,
                                          action = "replace",
                                          candidates = candidates,
                                          landscape = bathymetry)
## End(Not run)
```

---

make.windfarms *Make wind farm construction scenario*

Description

Produce a hypothetical wind farm construction scenario, specifying the position and timing of individual piling events, as well as the sound source level. All wind farms are assumed to consist of the same number of turbines, laid out in a rectangular grid. The start and end tick (i.e. the number of half-hour intervals since simulation start) is generated based on provided values for the time it required for each piling and the time between piling events.

Usage

```
make.windfarms(
  area.file,
  area.def,
  n.wf,
  n.turb,
  turb.dist,
  min.wf.dist,
  impact,
  constr.start,
  constr.end,
  constr.time,
  constr.break,
  iterate = 10000,
  verbose = FALSE,
  wf.coords = "random"
)
```

Arguments

area.file	Name of the raster file specifying where the wind farms should be constructed.

area.def	Value in area.file for the areas were wind farms can be located
n.wf	Number of wind farms to construct
n.turb	Total number of turbines to construct
turb.dist	Distance between turbines within a wind farm (meters)
min.wf.dist	Minimum distance between wind farms (meters)
impact	Sound source level (dB); sound emitted from turbines during construction, i.e. from tickStart to tickEnd (including both start and end)
constr.start	The tick at which construction of the first turbine starts.
constr.end	The tick at which construction of the very last turbine in the last wind farm ends.
constr.time	The time it takes to construct a single wind turbine (number of ticks).
constr.break	Break between individual pilings within a wind farm, counted in number of half-hour 'ticks'.
iterate	Number of times to try finding a spot for a new wind farm that is sufficiently far from the nearest neighbouring wind farm (>min.wf.dist). The number also defines the number of random positions to search through.
verbose	Logical; whether messages should be printed to console.
wf.coords	Possible location of the south-western corner of the wind farms. Defaults to the text "random", but can also be a data frame with coordinates in the columns x and y.

Value

data.frame specifying the position of each turbine in a wind farm, along with the start time and end time for pile driving of the turbine foundation and the sound source level during pile driving. Can be exported as a text file and used for controlling DEPONS simulations.

Note

The parameters constr.start, constr.end, constr.time, and constr.break are truncated to nearest integer value. Construction of wind farms starts in WF001 at tick constr.start. Each turbine foundation is piled over a period of constr.time, followed by a noise-free period of constr.break. Several pile driving operations may take place at the same time, to ensure that the last piling ends before constr.end.

---

plot,DeponsBlockdyn,missing-method

Plot a DeponsBlockdyn object

Description

Plot population dynamics simulated with DEPONS

Usage

```
## S4 method for signature 'DeponsBlockdyn,missing'
plot(x, y, dilute = 5, ...)
```

Arguments

x	DeponsBlockdyn object
y	Not used
dilute	Integer. Plot only one in every 'dilute' values. Defaults to 5, which yields a plot of the first simulated value and one in every five of the following values.
...	Optional plotting parameters

Value

`data.frame` listing blocks where no animals were counted (returned invisibly)

Note

The function returns a data frame with numbers of blocks with no agents.

Examples

```
data("porpoisebdyn")
my.col <- c("red", "darkgreen", "orange")
plot(porpoisebdyn, col=my.col)
legend("bottomright", bty="n", fill=my.col, legend=paste("Block", 0:2))

# Show all data points for small range of x-values
plot(porpoisebdyn, xlim=c(1950, 2050), ylim=c(4850, 5050), type="p", dilute=1, col=my.col)
```

`plot,DeponsDyn,missing-method` *Plot a DeponsDyn object*

Description

Plot population dynamics simulated with DEPONS

Usage

```
## S4 method for signature 'DeponsDyn,missing'
plot(x, y, dilute = 5, plot.energy = TRUE, plot.legend = TRUE, ...)
```

Arguments

x	DeponsDyn object
y	Not used
dilute	Integer. Plot only one in every 'dilute' values. Defaults to 5, which yields a plot of the first simulated value and one in every five of the following values.
plot.energy	If set to TRUE it plots the amount of energy stored in simulated and in the landscape in addition to the population count
plot.legend	If set to TRUE, a legend is plotted
...	Optional plotting parameters

Examples

```
data("porpoisedyn")

# Plot for specific range of years
rg <- c(as.POSIXlt("2011-01-01", tz = "UTC"), as.POSIXlt("2018-12-31", tz = "UTC"))
plot(porpoisedyn, xlim=as.POSIXct(rg), plot.energy=TRUE)

## Not run:
# Read data from default DEPONS simulation directory:
sim.dir <- "/Applications/DEPONS 2.1/DEPONS"
new.sim.name <- get.latest.sim(dir=sim.dir)
new.sim.out <- read.DeponsDyn(fname=paste(sim.dir, new.sim.name, sep="/"))
plot(new.sim.out)

## End(Not run)
```

---

plot,DeponsRaster,ANY-method

Plot a DeponsRaster object

Description

Plot the values in a DeponsRaster object. Porpoisetracks or other kinds of lines, poits etc. can be drawn on top of the plot by adding

Usage

```
## S4 method for signature 'DeponsRaster,ANY'
plot(x, y, col, trackToPlot = 1, ...)
```

Arguments

x	DeponsRaster object
y	A DeponsTrack object or missing

col	A color palette, i.e. a vector of n contiguous colors. Reasonable defaults are provided.
trackToPlot	Integer indicating which track to plot if the DeponsTrack object contains more than one track. Ignored if y is missing
...	Other optional plotting parameters, including 'axes', 'legend', and 'main'.

Value

No return value, called for side effects

See Also

See method for plot in the raster package for plotting parameters and plot.DeponsTrack for plotting of DeponsRasters cropped to the extent of tracks.

---

plot,DeponsShips,missing-method

Plot a DeponsShips object

Description

Plot the tracks that ship agents move along in DEPONS.

Usage

```
## S4 method for signature 'DeponsShips,missing'
plot(x, y, ...)
```

Arguments

x	DeponsShips object
y	Not used
...	Optional plotting parameters, including 'col', 'main', 'add.legend', and 'legend.xy' (defaults to 'topright' when add.legend=TRUE)

Value

No return value, called for side effects

Examples

```
data(shipdata)
plot(shipdata, col=c("red", "green", "blue"))


# convert route coordinate units from 'grid squares' to UTM
data(bathymetry)
out <- summary(bathymetry)
left <- out[[4]][1]
bottom <- out[[4]][2]
for (i in 1:3) {
    newroute <- shipdata@routes[[2]][[i]]*400
    newroute$x <- newroute$x + as.numeric(left)
    newroute$y <- newroute$y + as.numeric(bottom)
    shipdata@routes[[2]][[i]] <- newroute
    }

# Reproject coastline and clip to size of Kattegat landscape
library(sp)
data(bathymetry)
data(coastline)
coastline_sf <- sf::st_as_sf(coastline)
coastline2 <- sf::st_transform(coastline_sf, crs(bathymetry))
bbox <- bbox(bathymetry)
clip.poly <- make.clip.poly(bbox, crs(bathymetry))
plot(shipdata, col=c("red", "green", "blue"), add=TRUE, add.legend=TRUE)
plot(clip.poly, add=TRUE)
```

---

plot,DeponsTrack,missing-method

Plot a DeponsTrack object

Description

Plot the coordinates in a movement track simulated with DEPONS.

Usage

```
## S4 method for signature 'DeponsTrack,missing'
plot(x, y, trackToPlot = 1, add = FALSE, ...)
```

Arguments

x	DeponsTrack object
y	Not used
trackToPlot	Integer; indicates which track to plot if there is more than one track in the object. Defaults to 1

add	Logical, whether to add the track to an existing plot one animal was tracked during the simulation.
...	Optional plotting parameters

Value

No return value, called for side effects

Examples

```
data(porpoisetrack)
data("porpoisetrack")
plot(porpoisetrack)
```

`plot_calib02` *Plotting of simulated fine or large-scale metrics against argos data*

Description

Plotting of simulated fine or large-scale metrics against argos data

Usage

```
plot_calib02(sim.metrics, option)
```

Arguments

sim.metrics	A dataframe of movement metrics obtained with the calib_02 function
option	A character string, either ‘"fine"‘ or ‘"large"‘, indicating which set of Argos metrics (fine-scale or large-scale) to plot against simulated ones.

Value

a plot of real (Argos) vs simulated metrics

porpoisebdyn *Simulated porpoise population dynamics*

Description

An object of class DeponsBlockdyn with output from a DEPONS simulation based on the North Sea landscape, using a landscape divided into two blocks. Numbers of animals are counted per block.

Format

DeponsBlockdyn

See Also

DeponsBlockdyn-class, porpoisedyn

porpoisedyn *Simulated porpoise population dynamics*

Description

An object of class DeponsDyn with output from a DEPONS simulation based on the Kattegat landscape, assuming that the simulation represents the period 2010-01-01 onward in the real world. Number of animals and energy availability is recorded for the entire landscape.

Format

DeponsDyn

See Also

DeponsDyn-class, porpoisebdyn

porpoisetrack *Simulated porpoise track*

Description

An object with five elements: `title`, `landscape`, `simtime`, `crs`, and `tracks`. The `crs` stores information about the map projection used ("+proj=utm +zone=32 +datum=WGS84 +units=m +no_defs"). The `tracks` element is a list of objects of class `SpatialPointsDataFrame`, each ofwhich corresponds to one simulated animal. `simtime` is the simulation date.

Format

DeponsTrack

See Also

`DeponsTrack-class`

read.DeponsBlockdyn *Reading simulated population count for blocks*

Description

Function for reading DEPONS simulation output with number of animals per block for each time step.

Usage

```
read.DeponsBlockdyn(
  fname,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  timestep = 30,
  startday = "2010-01-01",
  tz = "UTC"
)
```

Arguments

`fname`	Name of the file (character) that contains movement data generated by DEPONS. The name includes the path to the directory if this is not the current working directory.
`title`	Optional character string giving name of simulation
`landscape`	The landscape used in the simulation

simtime	Optional text string with date of simulation (format: yyyy-mm-dd). If not provided this is obtained from name of input file
timestep	Time step used in the model, in minutes. Default 30 minutes
startday	The start of the period that the simulation represents, i.e. the real-world equivalent of 'tick 1' (character string of the form 'yyyy-mm-dd', or POSIXlt). Default "2010-01-01"
tz	Time zone. In DEPONS times are generally assumed to be in "UTC" (Coordinated Universal Time).

Value

DeponsBlockdyn object

See Also

See DeponsBlockdyn-class for details on what is stored in the output object and read.DeponsParam for reading the parameters used in the simulation.

Examples

```
## Not run:
# File loaded from default location
the.file <- "/Applications/DEPONS 2.1/DEPONS/PorpoisePerBlock.2020.Sep.02.20_24_17.csv"
file.exists(the.file)
porpoise.blockdyn <- read.DeponsBlockdyn(fname=the.file,
  title="Test simulation with two blocks", landscape="North Sea")
porpoise.blockdyn

# Get the latest simulation
the.file <- get.latest.sim(type="blockdyn", dir="/Applications/DEPONS 2.1/DEPONS")
owd <- getwd()
setwd("/Applications/DEPONS 2.1/DEPONS")
porpoise.blockdyn <- read.DeponsBlockdyn(fname=the.file)
setwd(owd)

## End(Not run)
```

---

read.DeponsDyn *Reading DEPONS simulation output*

Description

Function for reading simulation output produced by DEPONS.

Usage

```
read.DeponsDyn(
  fname,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  startday = "2010-01-01",
  timestep = 30,
  tz = "UTC"
)
```

Arguments

fname	Name of the file (character) that contains number of animals for each time step during the simulation, along with their energy and the amount of food in the landscape. The name includes the path to the directory if this is not the current working directory.
title	Optional character string giving name of simulation
landscape	The landscape used in the simulation
simtime	Optional character string with the date and time when the simulation finished (format yyyy-mm-dd). If not provided this is obtained from name of input file
startday	The start of the period that the simulation represents, i.e. the real-world equivalent of 'tick 1' (character string of the form 'yyyy-mm-dd', or POSIXlt). Default "2010-01-01"
timestep	Time step used in the model, in minutes. Default 30 minutes.
tz	Time zone. In DEPONS times are generally assumed to be in "UTC" (Coordinated Universal Time).

Value

DeponsDyn object containing simulation output

See Also

See DeponsDyn-class for details on what is stored in the output object and as.data.frame for converting from data frame.

Examples

```
## Not run:
dyn.file <- "/Applications/DEPONS 2.1/DEPONS/Statistics.2020.Sep.02.20_24_17.csv"
file.exists(dyn.file)
porpoisedyn <- read.DeponsDyn(dyn.file, startday=as.POSIXlt("2010-01-01", tz = "UTC"))
porpoisedyn

## End(Not run)
```

`read.DeponsDynBatch` *Read and merges DEPONS Batchmap and Statistics Files*

Description

Reads batch map files and statistics files from a specified directory and returns a list of 'DeponsDyn' objects and parameter values.

Usage

```
read.DeponsDynBatch(
  dir,
  par,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  startday = "NA",
  timestep = 30,
  tz = "UTC"
)
```

Arguments

`dir`	Character string specifying the directory path containing the 'Batchmap' and 'Statistics' files.
`par`	Character vector specifying the column names to extract from the batch map file for each run. #' These parameters are then stored in the 'Parameters' list.
`title`	Optional character string
`landscape`	Character string. Name of the simulation landscape. Default is "NA".
`simtime`	Optional character string with the date and time when the simulation finished (format yyyy-mm-dd).
`startday`	The start of the period that the simulation represents, i.e. the real-world equivalent of 'tick 1' (character string of the form 'yyyy-mm-dd', or POSIXlt)
`timestep`	Time step used in the model, in minutes. Defaults to 30 in DEPONS.
`tz`	Timezone.

Value

A list of 'DeponsDyn' objects and parameter values associated with run id

Examples

```
## Not run:
# Specify the directory containing Batchmap and Statistics files
dir_path <- "path/to/batchdata"
```

```
# Specify parameters to extract from Batchmap files
par <- c("parameter1", "parameter2")

# Run the function
results <- read.DeponsBatch(
  dir = dir_path,
  par = par,
  startday = "2010-01-01"
)

## End(Not run)
```

`read.DeponsParam` *Read simulation parameters*

Description

Read the parameters that were used for running a specific DEPONS simulation

Usage

```
read.DeponsParam(fname)
```

Arguments

`fname`	Name of the XML file (character) that contains the parameter list used for running a DEPONS simulation. The name includes the path to the directory if this is not the current working directory.

Details

The parameter file can be generated from within DEPONS by pressing the 'Save' icon after modifying the user settings on the 'Parameters' tab within the main DEPONS model window. See TRACE document for details regarding the parameters in the model: https://github.com/jacobnabe/DEPONS. It is strongly recommended that the parameter list is stored with the simulation output.

Value

Data frame containing all parameters used in a specific simulation

Examples

```
## Not run:
# Parameters read from file created by DEPONS run in interactive mode
the.file <- "/Applications/DEPONS 2.1/DEPONS/DEPONS.rs/parameters.xml"
pfile <- read.DeponsParam(the.file)

## End(Not run)
```

`read.DeponsRaster` *Reading DEPONS raster files*

Description

Function for reading raster files that have been used in DEPONS simulations. DEPONS rasters define amount of food available for simulated animals, spatial distribution of food patches, bathymetry, and distance to coast (dtc). The 'blocks' raster enables the user to count animals in specific parts of the landscape during simulations. See Nabe-Nielsen et al. (2018) for details regarding these files. In DEPONS 2.0 the salinity raster file was introduced; see TRACE document for details: https://github.com/jacobnabe/DEPONS

Usage

```
read.DeponsRaster(fname, type = "NA", landscape = "NA", crs = "NA")
```

Arguments

`fname`	Filename (character), including the path to the DEPONS raster file.
`type`	The kind of data stored in the raster; c('food', 'patches', 'bathymetry', 'dtc', 'salinity', 'blocks').
`landscape`	Identifier for the landscape used in the DEPONS simulations; typically set to 'North Sea'.
`crs`	CRS-object providing the map projection (see CRS).

Value

Returns a DeponsRaster object. The object inherits slots from the "RasterLayer" class, including "title", which is used for storing the file name.

References

Nabe-Nielsen, J., van Beest, F. M., Grimm, V., Sibly, R. M., Teilmann, J., & Thompson, P. M. (2018). Predicting the impacts of anthropogenic disturbances on marine populations. Conservation Letters, 11(5), e12563. doi:10.1111/conl.12563

See Also

`DeponsRaster-class`

read.DeponsShips *Read DEPONS ship files*

Description

Function for reading the json-files that are used for controlling how ship agents behave in DEPONS. Ships move along pre-defined routes in 30-min time steps. The routes are defined by the fix-points provided in the json file, and the geographic projection is assumed to match that of the landscape.

Usage

```
read.DeponsShips(fname, title = "NA", landscape = "NA", crs = as.character(NA))
```

Arguments

fname	Name of the file (character) where ship routes and ships are defined.
title	Optional character string with the name of the simulation
landscape	Optional character string with the landscape used in the simulation
crs	Character, coordinate reference system (map projection)

Value

Returns an object with the elements title landscape, crs, routes and ships.

See Also

ais.to.DeponsShips, write.DeponsShips

read.DeponsTrack *Reading DEPONS track files*

Description

Function for reading movement tracks produced by DEPONS. These describe movements of simulated animals within the simulation landscape, where the positions after each 30-min time step are provided using the coordinate reference system that were used for generating these landscapes.See van Beest et al. (2018) and Nabe-Nielsen et al. (2013) for details regarding how these files were generated as a balance between correlated random walk behaviour and spatial memory behaviour, which allows animals to return to previously visited food patches.

Usage

```
read.DeponsTrack(
  fname,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  crs = as.character(NA),
  tz = "UTC"
)
```

Arguments

fname	Name of the file (character) that contains movement data generated by DEPONS. The name includes the path to the directory if this is not the current working directory.
title	Optional character string giving name of simulation
landscape	Optional character string with the landscape used in the simulation
simtime	Character sting with date of simulation (format yyyy-mm-dd). If not provided this is obtained from name of input file
crs	Character, coordinate reference system (map projection)
tz	Time zone used in simulations. Defaults to UTC. #'

Value

Returns a DeponsTrack object with the elements title, simtime, crs, and tracks. The date is extracted from input data if not provided explicitly and stored as a POSIXlt object. The element tracks is a list of objects of class SpatialPointsDataFrame, each of which corresponds to one simulated animal (several animals can be tracked in one simulation).

Examples

```
data(porpoisetrack) # Load data for use in example

# Use standard DEPONS coordinate reference system / map projection:
the.crs <- "+proj=laea +lat_0=52 +lon_0=10 +x_0=4321000 +y_0=3210000
 +datum=WGS84 +units=m +no_defs"

## Not run:
one.fname <- "~/Applications/DEPONS/
  RandomPorpoise.2020.Jul.31.09_43_10.csv"

porpoisetrack <- read.DeponsTrack(one.fname, title="Track simulated using DEPONS 2.0",
  crs=the.crs)


## End(Not run)

# Plot the first of the simulated tracks
plot(porpoisetrack)
```

`read.DeponsTrackBatch` *Read and Process DEPONS Batchmap and Statistics Files*

Description

Reads batch map files and random porpoise files from a specified directory, merges them for each run, and returns a list of 'DeponsTrack' objects and parameter values

Usage

```
read.DeponsTrackBatch(
  dir,
  par,
  title = "NA",
  landscape = "NA",
  simtime = "NA",
  crs = as.character(NA),
  tz = "UTC"
)
```

Arguments

`dir`	Character string specifying the directory path containing the 'Batchmap' and 'RandomPorpoise' files
`par`	Character vector specifying the column names to extract from the batch map file for each run. These parameters are then stored in the 'Parameters' list
`title`	Optional character string giving name of simulation
`landscape`	Character string. Name of the simulation landscape
`simtime`	Character sting with date of simulation (format yyyy-mm-dd). If not provided this is obtained from name of input file
`crs`	Character, coordinate reference system (map projection)
`tz`	Time zone used in simulations. Defaults to UTC/GMT

Value

A list of 'DeponsTrack' objects and parameter values associated with run id

Examples

```
## Not run:
# Specify the directory containing Batchmap and Statistics files
dir_path <- "path/to/batchdata"

# Specify parameters to extract from Batchmap files
par <- c("parameter1", "parameter2")
```

```
# Run the function
results <- read.DeponsBatch(
  dir = dir_path,
  par = par,
  crs = "+proj=longlat +datum=WGS84"
)

## End(Not run)
```

`routes` *Get or define routes in DeponsShips objects*

Description

Get or define routes in DeponsShips objects

Usage

```
## S4 method for signature 'DeponsShips'
routes(x)

## S4 replacement method for signature 'DeponsShips'
routes(x) <- value
```

Arguments

`x`	Object of class `DeponsShips`
`value`	list with one named element per shipping route. Each element is a data frame with the variables x, y, speed, and 'pause' which define the coordinates of the fix-points on the shipping routes and the speeds that ships have after passing the fix point and until reaching the next fix point. The variable 'pause' instructs ships about how many minutes to wait before continuing to move.

Note

The unit of 'speed' is knots.

See Also

`ships`

`set.ship.type` *Convert AIS vessel type identifiers into the ship types recognized by DEPONS*

Description

Processes a `DeponsShips` object to convert numerical (and some written-out) AIS vessel type identifiers into the types recognized by DEPONS, if the correct types have not already been provided when generating the object (see `ais.to.DeponsShips`). Type assignment follows MacGillivray & de Jong (2021) and is augmented in some cases by the ship's length and calculated speed.

Usage

```
set.ship.type(data, list.ur = FALSE)
```

Arguments

`data`	A `DeponsShips` object
`list.ur`	If true, will print a message when a ship type identifier is not recognized and set to the catch-all of "Other". Defaults to FALSE.

Details

The safe way to identify ship type is by numeric AIS code - all numbers are handled. Not all of the written-out type names and none of the detailed type names (Standby Safety, Fire Ship, etc.) are recognized, and these instances will be recoded to "Other". Note that ships in the "Other" category are also considered as candidates for the assignment of active pauses using `make.stationary.ships` (e.g., crew transfer vessels and offshore supply ships), thus it is recommended to avoid overloading the category with random vessels.

Value

A `DeponsShips` object with ship types that are recognized by DEPONS

Reference

MacGillivray, A., & de Jong, C (2021). A reference spectrum model for estimating source levels of marine shipping based on Automated Identification System data. Journal of Marince Science and Engineering, 9(4), 369. doi:10.3390/jmse9040369

Examples

```
data(shipdata)
shipdata@ships$type <- c(60,99,"Cruise",60,55,"Dredger",60,33,33,31,60,60,60,60,"Cruise")
ships(shipdata)
shipdata <- set.ship.type(shipdata)
ships(shipdata)
```

shipdata *Ships on example routes through the Kattegat*

Description

The routes of fifteen ships of different types in the Kattegat during a period of 15 days. The fix points that define the routes use the UTM zone 32 projection (CRS = "+proj=utm +zone=32 +units=m +no_defs +datum=WGS84"; EPSG:32632; see https://epsg.io/32632).

Format

DeponsShips object

See Also

[DeponsShips-class]

ships *Get or define ships in DeponsShips objects*

Description

Get or define ships in DeponsShips objects

Usage

```
## S4 method for signature 'DeponsShips'
ships(x)

ships(x) <- value
```

Arguments

x	Object of class DeponsShips
value	data frame with the 'name', 'type', 'length', and 'route' of ships to be simulated, as well as 'tickStart' and 'tickEnd' defining when the ships are to be included in simulations. 'route' is one of the shipping routes defined in the DeponsShips object.

See Also

routes

Examples

```
data(shipdata)
ships(shipdata)
```

startday *Get or set start date for simulation*

Description

Get or set start date for simulation

Get or set start date for simulation

Usage

```
## S4 method for signature 'DeponsBlockdyn'
startday(x)

## S4 method for signature 'DeponsDyn'
startday(x)

## S4 replacement method for signature 'DeponsBlockdyn'
startday(x) <- value

## S4 replacement method for signature 'DeponsDyn'
startday(x) <- value
```

Arguments

x	Object of class DeponsDyn
value	POSIXlt or character string of the form 'yyyy-mm-dd'

Details

The start date indicates the start of the period that the simulation is supposed to represent.

The start date indicates the start of the period that the simulation is supposed to represent.

Note

The assignment of a new start time is currently quite time consuming.

Summary-methods *Summary*

Description

Summarizes different kinds of objects created based on output from the DEPONS model

Usage

```
## S4 method for signature 'DeponsBlockdyn'
summary(object)

## S4 method for signature 'DeponsDyn'
summary(object)

## S4 method for signature 'DeponsRaster'
summary(object)

## S4 method for signature 'DeponsShips'
summary(object)

## S4 method for signature 'DeponsTrack'
summary(object)
```

Arguments

object	Depons* object

Details

The summary method is available for DeponsTrack-class, DeponsDyn-class, DeponsRaster-class, and DeponsBlockdyn-class-objects.

Value

list summarizing the DeponsBlockdyn object

table summarizing the DeponsBlockdyn object

list summarizing the DeponsRaster object

list summarizing the DeponsTrack object

---

tick.to.time *Convert tick number to time object*

Description

Converts the number of ticks since the start of the simulation to a specific date while taking into account that DEPONS assumes that there are 360 days in a simulation year.

Usage

```
tick.to.time(tick, timestep = 30, origin = "2010-01-01", tz = "UTC", ...)
```

Arguments

tick	Numeric, or numeric vector; tick number
timestep	Numeric; length of each simulation time step, in minutes. Defaults to 30 minutes.
origin	Character. The first day of the period that the simulation represents, format: 'yyyy-mm-dd'.
tz	Character. Valid time zone code (default UTC).
...	Optional parameters

Value

object of class as.POSIXlt

Note

The function assumes that there are 30 days in each month, except in January, February and March with 31, 28 and 31 days, respectively.

See Also

time.to.tick is the inverse of this function, converting dates to ticks

---

time.to.tick *Convert date to tick number*

Description

Convert a date to the number of ticks since simulation start while taking into account that DEPONS assumes that there are 360 days in a simulation year.

Usage

```
time.to.tick(time, timestep = 30, origin = "2010-01-01", tz = "UTC", ...)
```

Arguments

time	Character, or character vector, of the form 'YYYY-MM-DD' (or 'YYYY-MM-DD HH:MM:SS'), or equivalent POSIX object. Date(s) to be converted to ticks.
timestep	Numeric (default 30). Length of each simulation time step in minutes.
origin	Character of the form 'YYYY-MM-DD' or equivalent POSIX object (default "2010-01-01"). Start date of simulation.
tz	Character. Valid time zone code (default UTC).
...	Optional parameters.

Details

Times are rounded down to the current 30-minute interval during conversion. The function assumes that there are 30 days in each month, except in January, February and March with 31, 28 and 31 days, respectively. Provided dates that fall on days that are not accommodated (February 29, and the 31st day of the months May, July, August, October, and December) are returned as NA.

The function may be used to, e.g., convert recorded piling dates to ticks for use in wind farm scenarios (see `make.windfarms` for construction of hypothetical scenarios from parametric inputs).

Value

Numeric vector of tick numbers.

See Also

`tick.to.time` is the inverse of this function, converting ticks to dates

Examples

```
## Not run:
#Uses date column of AIS data.
#Times are in 30-minute intervals, and converting back yields the same times
data(aisdata)
ticks <- time.to.tick(aisdata$time, origin = "2015-12-20")
times_reconverted <- tick.to.time(ticks, origin = "2015-12-20")

#Uses dates at other intervals.
#Converting back yields times rounded down to the current 30-minute interval
times <- c("2016-12-20 00:10:00",
          "2016-12-20 02:45:30",
          "2016-12-20 05:01:05",
          "2016-12-22 01:30:00")
ticks <- time.to.tick(times, origin = "2015-12-20")
times_reconverted <- tick.to.time(ticks, origin = "2015-12-20")
## End(Not run)
```

---

title<- *Get or set the title of Depons* objects*

Description

Get or set the title of Depons* objects

Usage

```
## S4 replacement method for signature 'DeponsTrack'
title(x) <- value

## S4 replacement method for signature 'DeponsDyn'
title(x) <- value

## S4 replacement method for signature 'DeponsShips'
title(x) <- value

## S4 method for signature 'DeponsTrack'
title(x, value)

## S4 method for signature 'DeponsDyn'
title(x, value)

## S4 method for signature 'DeponsShips'
title(x, value)
```

Arguments

x	Object of class DeponsTrack, DeponsDyn, DeponsBlockdyn or DeponsShips
value	Character string

---

write,DeponsShips-method

Write DEPONS ship files

Description

Function for writing a json-file for controlling how ship agents behave in DEPONS. Ships move along pre-defined routes in 30-min time steps. The routes are defined by the fix-points provided in the json file, and the geographic projection is assumed to match that of the landscape. The projection is not stored as part of the json file.

Usage

```
## S4 method for signature 'DeponsShips'
write(x, file)
```

Arguments

x	Name of the DeponsShips object to be exported
file	Name of the output file (character)

Value

No return value, called for side effects

Note

The exported json file is intended for use in DEPONS 2.3 or later (released July 2022) where the sound pressure level (SPL) is calculated within DEPONS based on ship type, ship length and speed.

Index

∗ **datasets**
aisdata, 4
argosmetrics, 5
bathymetry, 8
coastline, 13
porpoisebdyn, 40
porpoisedyn, 40
porpoisetrack, 41

ais.to.DeponsShips, 3, *12*, *20*, *26–28*, *31*, *32*, *47*, *51*
aisdata, *4*, 4
argosmetrics, 5
as.data.frame, *43*
as.data.frame,DeponsDyn-method, 5
as.data.frame,DeponsTrack-method, 6
as.DeponsRaster, 7, *8*
as.DeponsRaster,RasterLayer-method *(*as.DeponsRaster*)*, 7
as.POSIXct, *20*
as.POSIXlt, *55*
as.raster, *7*, 7
as.raster,DeponsRaster-method *(*as.raster*)*, 7

bathymetry, 8, *16*
bbox, 9, *25*
bbox,DeponsRaster-method *(*bbox*)*, 9
bbox,DeponsTrack-method *(*bbox*)*, 9

calib_01, 9
calib_02, 10
check.DeponsShips, *4*, 11, *27*
coastline, 13
CRS, *17*, *46*
crs, 13
crs,DeponsRaster-method *(*crs*)*, 13
crs,DeponsShips-method *(*crs*)*, 13
crs,DeponsTrack-method *(*crs*)*, 13
crs<- *(*crs*)*, 13
crs<-,DeponsRaster-method *(*crs*)*, 13
crs<-,DeponsShips-method *(*crs*)*, 13
crs<-,DeponsTrack-method *(*crs*)*, 13

DEPONS2R, 14
DeponsBlockdyn, *14*
DeponsBlockdyn-class, 14
DeponsDyn, *14*
DeponsDyn-class, 15
DeponsRaster, *14*
DeponsRaster-class, 16
DeponsShips, *14*
DeponsShips-class, 17
DeponsTrack, *14*
DeponsTrack-class, 17
dyn, 18
dyn,DeponsBlockdyn-method *(*dyn*)*, 18
dyn,DeponsDyn-method *(*dyn*)*, 18

get.latest.sim, 18, *19*
get.simtime, 19

interpolate.ais.data, *4*, 20
interpolate.maps, 21

landscape *(*landscape<-*)*, 23
landscape,DeponsBlockdyn-method *(*landscape<-*)*, 23
landscape,DeponsDyn-method *(*landscape<-*)*, 23
landscape,DeponsTrack-method *(*landscape<-*)*, 23
landscape<-, 23
landscape<-,DeponsBlockdyn-method *(*landscape<-*)*, 23
landscape<-,DeponsDyn-method *(*landscape<-*)*, 23
landscape<-,DeponsTrack-method *(*landscape<-*)*, 23

make.blocksraster, *16*, 23

make.blocksraster,DeponsRaster-method (make.blocksraster), 23
make.clip.poly, *9*, 25
make.clip.poly,matrix-method (make.clip.poly), 25
make.construction.traffic, 26
make.DeponsDyn, 29
make.stationary.ships, *27, 28*, 30, *51*
make.windfarms, *26–28*, 33, *56*

plot, *37*
plot,DeponsBlockdyn,missing-method, 34
plot,DeponsDyn,missing-method, 35
plot,DeponsRaster,ANY-method, 36
plot,DeponsRaster,DeponsTrack-method (plot,DeponsRaster,ANY-method), 36
plot,DeponsShips,missing-method, 37
plot,DeponsTrack,missing-method, 38
plot.DeponsBlockdyn, *15*
plot.DeponsBlockdyn (plot,DeponsBlockdyn,missing-method), 34
plot.DeponsDyn, *16*
plot.DeponsDyn (plot,DeponsDyn,missing-method), 35
plot.DeponsRaster, *16*
plot.DeponsRaster (plot,DeponsRaster,ANY-method), 36
plot.DeponsShips, *17*
plot.DeponsShips (plot,DeponsShips,missing-method), 37
plot.DeponsTrack, *18*, *37*
plot.DeponsTrack (plot,DeponsTrack,missing-method), 38
plot_calib02, 39
porpoisebdyn, *40*, 40
porpoisedyn, *40*, 40
porpoisetrack, 41
POSIXlt, *14, 15*, *19*, *48*

raster::raster, *21*
RasterLayer, *7*
read.DeponsBlockdyn, *15*, *19*, 41
read.DeponsDyn, *16*, 42
read.DeponsDynBatch, 44
read.DeponsParam, *42*, 45
read.DeponsRaster, *16*, 46
read.DeponsShips, *17*, *20*, 47
read.DeponsTrack, *18*, 47
read.DeponsTrackBatch, 49
routes, *4*, *17*, 50, *52*
routes,DeponsShips-method (routes), 50
routes<- (routes), 50
routes<-,DeponsShips-method (routes), 50

set.ship.type, 51
shipdata, 52
ships, *4*, *17*, *50*, 52
ships,DeponsShips-method (ships), 52
ships<- (ships), 52
ships<-,DeponsShips-method (ships), 52
SpatialPointsDataFrame, *18*, *41*, *48*
SpatialPolygonsDataFrame, *13*
st_crs, *18*
startday, 53
startday,DeponsBlockdyn-method (startday), 53
startday,DeponsDyn-method (startday), 53
startday<- (startday), 53
startday<-,DeponsBlockdyn-method (startday), 53
startday<-,DeponsDyn-method (startday), 53
summary (Summary-methods), 53
summary,DeponsBlockdyn-method (Summary-methods), 53
summary,DeponsDyn-method (Summary-methods), 53
summary,DeponsRaster-method (Summary-methods), 53
summary,DeponsShips-method (Summary-methods), 53
summary,DeponsTrack-method (Summary-methods), 53
Summary-methods, 53

tick.to.time, 54, *56*
time.to.tick, *26, 27*, *55*, 55
title, *4*
title (title<-), 56
title,DeponsDyn-method (title<-), 56
title,DeponsShips-method (title<-), 56
title,DeponsTrack-method (title<-), 56

title<-, 56
title<-,DeponsDyn-method (title<-), 56
title<-,DeponsShips-method (title<-), 56
title<-,DeponsTrack-method (title<-), 56

write,DeponsShips-method, 57
write.DeponsShips, 4, 47
write.DeponsShips
        (write,DeponsShips-method), 57