Package: birdscanR (via r-universe)

June 23, 2026

Title Migration Traffic Rate Calculation Package for 'Birdscan MR1' Radars

Version 0.3.0.9029

Description Extract data from 'Birdscan MR1' 'SQL' vertical-looking radar databases, filter, and process them to Migration Traffic Rates (# objects per hour and km) or density (# objects per km3) of, for example birds, and insects. Object classifications in the 'Birdscan MR1' databases are based on the dataset of Haest et al. (2021) <doi:10.5281/zenodo.5734960>). Migration Traffic Rates and densities can be calculated separately for different height bins (with a height resolution of choice) as well as over time periods of choice (e.g., 1/2 hour, 1 hour, 1 day, day/night, the full time period of observation, and anything in between). Two plotting functions are also included to explore the data in the 'SQL' databases and the resulting Migration Traffic Rate results. For details on the Migration Traffic Rate calculation procedures, see Schmid et al. (2019) <doi:10.1111/ecog.04025>.

License GPL (>= 3)

URL https://github.com/BirdScanCommunity/birdscanR, https://birdscancommunity.github.io/birdscanR/

BugReports https://github.com/BirdScanCommunity/birdscanR/issues

Depends R (>= 4.1.0)

Imports DBI, dplyr, lifecycle, ggplot2, methods, modi, reshape2, RODBC, RPostgreSQL, readr, rstudioapi, sp, stats, suntools, tibble, tidyr, utils, yaml

Suggests bioRad, knitr, rmarkdown

VignetteBuilder knitr

Encoding UTF-8

LazyData true

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libabsl-dev cmake libgdal-dev gdal-bin libgeos-dev libicu-dev unixodbc-dev libssl-dev libproj-dev libsqlite3-dev libudunits2-dev libx11-dev

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

Date/Publication 2026-06-23 15:49:06 UTC

RemoteUrl https://github.com/BirdScanCommunity/birdscanR

RemoteRef HEAD

RemoteSha c5a4b12a527d365b1f366d23732bf0c1d5761a5f

Contents

---

addDayNightInfoPerEcho *addDayNightInfoPerEcho*

Description

addDayNightInfoPerEcho() adds three columns 'dayOrNight', "dayOrCrepOrNight' and 'dateSunset' to the echo data. This allows the user to filter echo data easily by "day" and "night", or "day", "crepuscular", and "night".

Usage

```
addDayNightInfoPerEcho(
  echoData,
  sunriseSunset,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar"
)
```

Arguments

echoData	dataframe with the echo data from the data list created with extractDbData().
sunriseSunset	dataframe with sunrise/sunset and civil twilight times created with twilight()
sunOrCivil	optional character variable, Set to "sun" to use sunrise/sunset times or to "civil" to use civil twilight times to group echoes into day/night. Default is "civil".
crepuscule	optional character variable, Set to "nauticalSolar" to use the time between nautical dusk/dawn and sunrise/sunset times to define the crepuscular period, or to "nauticalCivil" to use the time between nautical and civil dusk/dawn to define the crepuscular period, or to "civilSolar" to use the time between civil dusk/dawn and sunrise/sunset times to define the crepuscular period. Default is "nauticalSolar".

Value

data frame with thre columns added, i.e. 'dayOrNight', 'dayOrCrepOrNight', and 'dateSunset'.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other manipulation functions: computeDensity(), computeMTR(), convertTimeZone(), createVPTS(), filterSpeedFeature37(), mergeVisibilityAndManualBlindTimes(), reclassToBats(), twilight()

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Add day/night info to echo data
# ======================================================================
echoData = addDayNightInfoPerEcho(
  echoData      = dbData$echoData,
  sunriseSunset = dbData$sunriseSunset,
  sunOrCivil    = "civil"
)
```

`CH_Sempach_2024_SEP24_25_DataExtract`
Example Birdscan MR1 data extract from Sempach, Switzerland

Description

A complete data extract from a Birdscan MR1 weather radar, as returned by `extractDbData()`. Collected at Sempach, Switzerland (47.13°N, 8.19°E) on September 24–25, 2024. Load with: `readRDS(system.file("extdata", "CH_Sempach_2024_SEP24_25_DataExtract.rds", package = "birdscanR"))`

Format

A named list as returned by `extractDbData()`, with components:

echoData data.frame with one row per detected echo

protocolData data.frame with protocol/collection intervals

siteData data.frame with radar site metadata

visibilityData data.frame with automatic visibility/blind times

timeBinData data.frame with raw time-bin table from the database

availableClasses character vector of classification labels

availableBatClasses character vector of bat classification labels

rfFeatures data.frame with RF feature metadata

TimeZone data.frame with radarTimeZone and targetTimeZone

classProbabilitiesAndMtrFactors data.frame with class probabilities

batProbabilitiesAndMtrFactors data.frame with bat class probabilities

sunriseSunset data.frame with sunrise/sunset and civil twilight times

See Also

Other sample data: classAbbreviations, loadManualBlindTimes(), manualBlindTimes

Examples

```
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
```

---

classAbbreviations *Default class abbreviations table of the birdscanR package*

Description

Table to allow for easy abbreviations of the standard classes of the Birdscan MR1.

Usage

```
data(classAbbreviations)
```

Format

An object of class data.frame.

See Also

Other sample data: CH_Sempach_2024_SEP24_25_DataExtract, loadManualBlindTimes(), manualBlindTimes

Examples

```
data(classAbbreviations)
```

---

compileData *compileData*

Description

compileData() filters database-extracts and save metadata used to compute MTR computeMTR(). It takes the output from extractDbData() and trunks the needed dataset to the restricted settings, e.g. time frame, pulse type.

Usage

```
compileData(
  echoData = NULL,
  protocolData = NULL,
  blindTimesData = NULL,
  sunriseSunsetData = NULL,
  radarSiteData = NULL,
  dbName = NULL,
  pulseTypeSelection = NULL,
  rotationSelection = NULL,
  timeRangeTargetTZ = NULL,
  targetTimeZone = "Etc/GMT0",
  classSelection = NULL,
  classProbCutOff = NULL,
  altitudeRange_AGL = NULL,
  echoValidator = FALSE,
  filePath = NULL,
  tagOutputFile = c(NULL, NULL),
  saveCSV = FALSE
)
```

Arguments

`echoData`	dataframe with the echo data from the data list created with `extractDbData()`.
`protocolData`	dataframe with the protocol data from the data list created with `extractDbData()`. Echoes not detected during the listed protocols will be excluded.
`blindTimesData`	dataframe with the manual blind times created by `loadManualBlindTimes()`. It include the automated blind times induced by changes in measurement protocol, and blind time added manually to remove periods of incoherent data collection.
`sunriseSunsetData`	dataframe with sunrise/sunset, and civil and nautical dawn/dusk. Computed with `twilight()`.
`radarSiteData`	dataframe/vector with the database site table.
`dbName`	Name of the database. Can be a useful meta data.
`pulseTypeSelection`	character vector with the pulse types which should be included in the subset. Options: `S`, `M`, `L`, i.e. short-, medium-, long-pulse, respectively. Default is NULL: no filtering applied based on pulseType.
`rotationSelection`	numeric vector to select the operation modes with and/or without antenna rotation. Options: 0, 1. (0 = no rotation, 1 = rotation). Default is NULL: no filtering applied based on rotation mode.
`timeRangeTargetTZ`	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M". Echoes outside the time range will be excluded.
`targetTimeZone`	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".

`classSelection` character string vector with the classes that should be included.

`classProbCutOff`
numeric cutoff value for class probabilities. Echoes with a lower class probability will be excluded.

`altitudeRange_AGL`
numeric vector of length 2 with start and end of the altitude range. Echoes outside the altitude range will be excluded.

`echoValidator` logical, If set to FALSE - default -, no additional filters is applied; if set to TRUE, echoes labelled by the echo validator as "non-bio scatterer" will be excluded.

`filePath` If given, the data-list is saved as RDS.

`tagOutputFile` Vector of two elements for prefix & suffix to file name, given `filePath` is not NULL.

`saveCSV` if true, save tables as CSV in a folder nested in `filePath`.

Value

Returns filtered data table - echo, protocol, blindTimes, sunriseSunset, radarSite - and necessary parameters as input for `computeMTR()`.

Author(s)

Baptiste Schmid, Fabian Hertner, Birgen Haest

See Also

Other write file functions: `saveMTR()`, `savePlotToFile()`

`computeDensity` *computeDensity*

Description

This function will estimate the density (expressed as #objects / km3) based on the observations in your database. Note that this function only works properly on Birdscan MR1 database versions >= 1.7.0.4 as the variable feature37.speed is required for the density calculation.

Usage

```
computeDensity(
  dbName,
  echoes,
  classSelection,
  altitudeRange,
  altitudeBinSize,
  timeRange,
  timeBinDuration_sec,
```

```
  timeZone,
  sunriseSunset,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar",
  protocolData,
  visibilityData,
  manualBlindTimes = NULL,
  saveBlindTimes = FALSE,
  blindTimesOutputDir = getwd(),
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0,
  computePerDayNight = FALSE,
  computePerDayCrepusculeNight = FALSE,
  computeAltitudeDistribution = TRUE
)
```

Arguments

dbName	Character string, containing the name of the database you are processing
echoes	dataframe with the echo data from the data list created with `extractDbData()` or a subset of it created by the function
classSelection	character string vector with all classes which should be used to calculate the density. The density and number of Echoes will be calculated for each class as well as for all classes together.
altitudeRange	numeric vector of length 2 with the start and end of the altitude range in meter a.g.l.
altitudeBinSize	numeric, size of the altitude bins in meter.
timeRange	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M"
timeBinDuration_sec	duration of timeBins in seconds (numeric). for values <= 0 a duration of 1 hour will be set
timeZone	time zone in which the time bins should be created as string, e.g. "Etc/GMT0"
sunriseSunset	dataframe with sunrise/sunset, and civil and nautical dawn/dusk. Computed with `twilight()`.
sunOrCivil	sunrise/sunset or civil dawn/dusk used to split day and night. Supported values: "sun" or "civil". Default: "civil"
crepuscule	optional character variable, Set to “nauticalSolar” to use the time between nautical dusk/dawn and sunrise/sunset times to define the crepuscular period, or to "nauticalCivil" to use the time between nautical and civil dusk/dawn to define the crepuscular period, or to "civilSolar" to use the time between civil dusk/dawn and sunrise/sunset times to define the crepuscular period. Default is "nauticalSolar".
protocolData	dataframe with the protocol data from the data list created by `extractDbData()` or a subset of it created by the function `filterProtocolData()`.

`visibilityData` dataframe with the visibility data from the data list created by `extractDbData()`.

`manualBlindTimes`
: dataframe with the manual blind times created by the function `loadManualBlindTimes()`.

`saveBlindTimes` Logical, determines whether to save the blind times to a file. Default: False.

`blindTimesOutputDir`
: Character string containing the path to save the blind times to. Default: 'your-working-directory'

`blindTimeAsMtrZero`
: character string vector with the blind time types which should be treated as observation time with MTR zero.

`propObsTimeCutoff`
: numeric between 0 and 1. Time bins with a proportional observation time smaller than `propObsTimeCutoff` are set to NA.

`computePerDayNight`
: logical, TRUE: density is computed per day and night. The time bins of each day and night will be combined and the mean density is computed for each day and night. The spread (first and third Quartile) for each day and night are also computed. The spread is dependent on the chosen time bin duration/amount of time bins; When FALSE: density is computed for each time bin. This option computes the density for each time bin defined in the time bin dataframe. The time bins that were split due to sunrise/sunset during the time bin will be combined to one bin.

`computePerDayCrepusculeNight`
: logical, TRUE: density is computed per crepusculeMorning, day, crepusculeEvening, and night. The time bins of each of these diel phases will be combined and the mean density is computed for each phase. The spread (first and third Quartile) for each phase is also computed. The spread is dependent on the chosen time bin duration/amount of time bins; When FALSE: density is computed for each time bin. This option computes the density for each time bin defined in the time bin dataframe. The time bins that were split due to sunrise/sunset during the time bin will be combined to one bin. Default = FALSE.

`computeAltitudeDistribution`
: logical, TRUE: compute the mean height and altitude distribution of density for the pre-defined quantiles 0.05, 0.25, 0.5, 0.75, 0.95

Value

Density

Author(s)

Birgen Haest, Fabian Hertner, Baptiste Schmid

See Also

Other manipulation functions: `addDayNightInfoPerEcho()`, `computeMTR()`, `convertTimeZone()`, `createVPTS()`, `filterSpeedFeature37()`, `mergeVisibilityAndManualBlindTimes()`, `reclassToBats()`, `twilight()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
dbName = "CH_Sempach_2024_SEP24_25"
targetTimeZone = "Etc/GMT0"
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")

# Set manual blind times to NULL (no manual blind times)
# ======================================================================
cManualBlindTimes = NULL

# Compute density
# ======================================================================
classSelection.density = c("passerine_type")
densityData = computeDensity(
  dbName = dbName,
  echoes = dbData$echoData,
  classSelection = classSelection.density,
  altitudeRange = c(25, 1025),
  altitudeBinSize = 50,
  timeRange = timeRangeData,
  timeBinDuration_sec = 1800,
  timeZone = targetTimeZone,
  sunriseSunset = dbData$sunriseSunset,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar",
  protocolData = dbData$protocolData,
  visibilityData = dbData$visibilityData,
  manualBlindTimes = cManualBlindTimes,
  saveBlindTimes = FALSE,
  blindTimesOutputDir = getwd(),
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0,
  computePerDayNight = FALSE,
  computePerDayCrepusculeNight = FALSE,
  computeAltitudeDistribution = TRUE
)
```

---

computeMTR *computeMTR*

Description

This function will estimate the Activity / Migration Traffic Rates (MTR, expressed as #objects / km / hour) based on the observations in your database.

Usage

```
computeMTR(
  dbName,
  echoes,
  classSelection,
  altitudeRange,
  altitudeBinSize,
  timeRange,
  timeBinDuration_sec,
  timeZone,
  sunriseSunset,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar",
  protocolData,
  visibilityData,
  manualBlindTimes = NULL,
  saveBlindTimes = FALSE,
  blindTimesOutputDir = getwd(),
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0,
  computePerDayNight = FALSE,
  computePerDayCrepusculeNight = FALSE,
  computeAltitudeDistribution = TRUE
)
```

Arguments

dbName	Character string, containing the name of the database you are processing
echoes	dataframe with the echo data from the data list created by `extractDbData()` or a subset of it created with `filterEchoData()`.
classSelection	character string vector with all classes which should be used to calculate the MTR. The MTR and number of Echoes will be calculated for each class as well as for all classes together.
altitudeRange	numeric vector of length 2 with the start and end of the altitude range in meter a.g.l.
altitudeBinSize	numeric, size of the altitude bins in meter.
timeRange	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M"
timeBinDuration_sec	duration of timeBins in seconds (numeric). for values <= 0 a duration of 1 hour will be set
timeZone	time zone in which the time bins should be created as string, e.g. "Etc/GMT0"
sunriseSunset	dataframe with sunrise/sunset, and civil and nautical dawn/dusk. Computed with `twilight()`.

`sunOrCivil` sunrise/sunset or civil dawn/dusk used to split day and night. Supported values: "sun" or "civil". Default: "civil"

`crepuscule` optional character variable, Set to "nauticalSolar" to use the time between nautical dusk/dawn and sunrise/sunset times to define the crepuscular period, or to "nauticalCivil" to use the time between nautical and civil dusk/dawn to define the crepuscular period, or to "civilSolar" to use the time between civil dusk/dawn and sunrise/sunset times to define the crepuscular period. Default is "nauticalSolar".

`protocolData` `data.frame` with the protocol data from the data list created by `extractDbData()` or a subset of it created with `filterProtocolData()`.

`visibilityData` dataframe with the visibility data from the data list created by `extractDbData()`.

`manualBlindTimes` dataframe with the manual blind times created by the function `loadManualBlindTimes()`.

`saveBlindTimes` Logical, determines whether to save the blind times to a file. Default: False.

`blindTimesOutputDir` Character string containing the path to save the blind times to. Default: 'your-working-directory'

`blindTimeAsMtrZero` character string vector with the blind time types which should be treated as observation time with MTR zero.

`propObsTimeCutoff` numeric between 0 and 1. Time bins with a proportional observation time smaller than `propObsTimeCutoff` are set to NA.

`computePerDayNight` logical, TRUE: MTR is computed per day and night. The time bins of each day and night will be combined and the mean MTR is computed for each day and night. The spread (first and third Quartile) for each day and night are also computed. The spread is dependent on the chosen time bin duration/amount of time bins; When FALSE: MTR is computed for each time bin. This option computes the MTR for each time bin defined in the time bin dataframe. The time bins that were split due to sunrise/sunset during the time bin will be combined to one bin.

`computePerDayCrepusculeNight` logical, TRUE: MTR is computed per crepusculeMorning, day, crepusculeEvening, and night. The time bins of each of these diel phases will be combined and the mean MTR is computed for each phase. The spread (first and third Quartile) for each phase is also computed. The spread is dependent on the chosen time bin duration/amount of time bins; When FALSE: MTR is computed for each time bin. This option computes the MTR for each time bin defined in the time bin dataframe. The time bins that were split due to sunrise/sunset during the time bin will be combined to one bin. Default = FALSE.

`computeAltitudeDistribution` logical, TRUE: compute the mean height and altitude distribution of MTR for the pre-defined quantiles 0.05, 0.25, 0.5, 0.75, 0.95

Value

Migration Traffic Rates

Author(s)

Fabian Hertner, Baptiste Schmid, Birgen Haest

See Also

Other manipulation functions: addDayNightInfoPerEcho(), computeDensity(), convertTimeZone(), createVPTS(), filterSpeedFeature37(), mergeVisibilityAndManualBlindTimes(), reclassToBats(), twilight()

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
dbName = "CH_Sempach_2024_SEP24_25"
targetTimeZone = "Etc/GMT0"
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")

# Set manual blind times to NULL (no manual blind times)
# ======================================================================
cManualBlindTimes = NULL

# Compute migration traffic rate
# ======================================================================
classSelection.mtr = c("passerine_type")
mtrData = computeMTR(
  dbName = dbName,
  echoes = dbData$echoData,
  classSelection = classSelection.mtr,
  altitudeRange = c(25, 1025),
  altitudeBinSize = 50,
  timeRange = timeRangeData,
  timeBinDuration_sec = 1800,
  timeZone = targetTimeZone,
  sunriseSunset = dbData$sunriseSunset,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar",
  protocolData = dbData$protocolData,
  visibilityData = dbData$visibilityData,
  manualBlindTimes = cManualBlindTimes,
  saveBlindTimes = FALSE,
  blindTimesOutputDir = getwd(),
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0,
  computePerDayNight = FALSE,
  computePerDayCrepusculeNight = FALSE,
  computeAltitudeDistribution = TRUE
)
```

convertTimeZone *Converts timestamps from radar time zone to an user-defined time zone*

Description

Converts timestamps from radar time zone to an user-defined time zone

Usage

```
convertTimeZone(
  data = NULL,
  colNames = "",
  originTZ = "Etc/GMT0",
  targetTZ = "Etc/GMT0"
)
```

Arguments

data	a data frame containing BirdScan data
colNames	a character vector containing valid column names, as present in data
originTZ	character, the time zone name of data to be converted (default is "etc/GMT0")
targetTZ	character, the time zone name to convert data into (default is "etc/GMT0")

Value

a data frame identical to data, any columns declared in colNames will have their name changed with a suffix (_originTZ or _targetTZ) added.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other manipulation functions: addDayNightInfoPerEcho(), computeDensity(), computeMTR(), createVPTS(), filterSpeedFeature37(), mergeVisibilityAndManualBlindTimes(), reclassToBats(), twilight()

Examples

```
# Load example data
# ================================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Add day/night info to echo data
# ================================================================================
echoData = convertTimeZone(
  data     = dbData$echoData,
  colNames = c("time_stamp"),
  originTZ = "Etc/GMT0",
  targetTZ = "Etc/GMT-2"
)
```

`createTimeRangeForPlot` *Create Time Range for Plot*

Description

Create a list by segmenting the input time range into regular periods for plots.

Usage

```
createTimeRangeForPlot(
  startDate = NULL,
  endDate = NULL,
  periodLength = 7,
  returnAsList = TRUE
)
```

Arguments

`startDate`	Per default, the first element of the input setting `timeRangeData`
`endDate`	Per default, the second element of the input setting ‘timeRangeData“
`periodLength`	Duration in days of each period
`returnAsList`	TRUE per default, otherwise as data.frame.

Value

A list of time periods

Author(s)

Baptiste Schmid, Birgen Haest

See Also

Other plot functions: plotExploration(), plotLongitudinalMTR()

Examples

```
## Not run:
# Set server, database, and other input settings
# ============================================================================
# Example with seven days time window
timeRangeData <- c("2024-01-01", "2024-02-15")
timeRangePlot <- createTimeRangePlot(timeRangeData[1], timeRangeData[2], 7)
print(timeRangePlot)

## End(Not run)
```

---

createVPTS *createVPTS*

Description

This function creates VPTS CSV output files in line with the ALOFT data standard, described here. Note that this function only works on Birdscan MR1 database versions >= 1.7.0.4 as the variable feature37.speed is required for the density calculation.

Usage

```
createVPTS(
  dbName,
  outputDir,
  echoes,
  classSelection = c("passerine_type", "wader_type", "swift_type", "large_bird",
    "unid_bird", "bird_flock"),
  altitudeRange = c(50, 1500),
  altitudeBinSize = 50,
  timeRange,
  timeBinDuration_sec = 900,
  timeZone = "Etc/GMT0",
  protocolData,
  visibilityData,
  siteData,
  sunriseSunset,
  manualBlindTimes = NULL,
  saveBlindTimes = FALSE,
```

```
  blindTimesOutputDir = getwd(),
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0.2
)
```

Arguments

dbName	Character string, containing the name of the database you are processing
outputDir	Character variable indicating where you want the VPTS files to be stored. The function will create a subdirectory called "vpts" within the specified outputDir.
echoes	dataframe with the echo data from the data list created with extractDbData() or a subset of it created by the function
classSelection	character string vector with all classes which should be used to calculate the density. The density and number of Echoes will be calculated for each class as well as for all classes together.
altitudeRange	numeric vector of length 2 with the start and end of the altitude range in meter a.g.l.
altitudeBinSize	numeric, size of the altitude bins in meter.
timeRange	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M"
timeBinDuration_sec	duration of timeBins in seconds (numeric). for values <= 0 a duration of 1 hour will be set
timeZone	time zone in which the time bins should be created as string, e.g. "Etc/GMT0"
protocolData	dataframe with the protocol data from the data list created by extractDbData() or a subset of it created by the function filterProtocolData().
visibilityData	dataframe with the visibility data from the data list created by extractDbData().
siteData	A data frame holding the site table, as extracted with extractDbData() or getSiteTable().
sunriseSunset	dataframe with sunrise/sunset, and civil and nautical dawn/dusk. Computed with twilight().
manualBlindTimes	dataframe with the manual blind times created by the function loadManualBlindTimes().
saveBlindTimes	Logical, determines whether to save the blind times to a file. Default: False.
blindTimesOutputDir	Character string containing the path to save the blind times to. Default: 'your-working-directory'
blindTimeAsMtrZero	character string vector with the blind time types which should be treated as observation time with MTR zero.
propObsTimeCutoff	numeric between 0 and 1. Time bins with a proportional observation time smaller than propObsTimeCutoff are set to NA.

Value

File path to the created VPTS CSV file.

Author(s)

Birgen Haest

See Also

Other manipulation functions: addDayNightInfoPerEcho(), computeDensity(), computeMTR(), convertTimeZone(), filterSpeedFeature37(), mergeVisibilityAndManualBlindTimes(), reclassToBats(), twilight()

Examples

```
# Load example data
# ============================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
dbName = "CH_Sempach_2024_SEP24_25"
mainOutputDir = tempdir()
targetTimeZone = "Etc/GMT0"
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")

# Set manual blind times to NULL (no manual blind times)
# ============================================================================
cManualBlindTimes = NULL

# Create vpts files
# ============================================================================
vptsDir = createVPTS(
  dbName = dbName,
  outputDir = mainOutputDir,
  echoes = dbData$echoData,
  altitudeRange = c(25, 1025),
  altitudeBinSize = 50,
  timeRange = timeRangeData,
  timeBinDuration_sec = 1800,
  timeZone = targetTimeZone,
  protocolData = dbData$protocolData,
  visibilityData = dbData$visibilityData,
  siteData = dbData$siteData,
  sunriseSunset = dbData$sunriseSunset,
  manualBlindTimes = cManualBlindTimes,
  saveBlindTimes = FALSE,
  blindTimesOutputDir = mainOutputDir,
  blindTimeAsMtrZero = NULL,
  propObsTimeCutoff = 0.2
)
```

dbConnectBirdscanSQL *Connect to a Birdscan SQL database*

Description

Connects to a Birdscan SQL database, be it a Microsoft SQL or postgreSQL database3

Usage

```
dbConnectBirdscanSQL(
  dbDriverChar = "SQL Server",
  dbServer = NULL,
  dbName = NULL,
  dbUser = NULL,
  dbPwd = NULL,
  dbPort = 5432
)
```

Arguments

dbDriverChar	'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'.
dbServer	NULL The name of the SQL Server. For a 'PostgreSQL' this can be a the host address or the internal IP.
dbName	NULL The name of the SQL database
dbUser	NULL The username of the SQL server
dbPwd	NULL The password for the user name
dbPort	5432 The dbPort parameter specifies the TCP/IP port number on which the PostgreSQL server is listening for connections (default is 5432)

Value

A database connection to your target database

Author(s)

Birgen Haest

See Also

Other read SQL database functions: QUERY(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
# Using and Microsoft SQL database
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ========================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ========================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

## End(Not run)
```

`extractDbData` *Extract DB Data*

Description

Load the data from the database or file and save it to file

Usage

```
extractDbData(
  dbDriverChar = "SQL Server",
  dbServer = NULL,
  dbName = NULL,
  dbUser = NULL,
  dbPwd = NULL,
  dbPort = 5432,
  saveDbToFile = FALSE,
  dbDataDir = NULL,
  radarTimeZone = NULL,
  targetTimeZone = "Etc/GMT0",
  timeInterval = NULL,
  listOfRfFeaturesToExtract = NULL,
```

```
  siteLocation = NULL,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar"
)
```

Arguments

dbDriverChar	'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'.
dbServer	NULL The name of the SQL Server. For a 'PostgreSQL' this can be a the host address or the internal IP.
dbName	NULL The name of the SQL database
dbUser	NULL The username of the SQL server
dbPwd	NULL The password for the user name
dbPort	5432 The dbPort parameter specifies the TCP/IP port number on which the PostgreSQL server is listening for connections (default is 5432)
saveDbToFile	FALSE Set to TRUE if you want to save the extracted database data to an rds file. The output filename is automatically set to dbName_DataExtract.rds
dbDataDir	NULL The path to the output directory where to store the extracted dataset. If the directory does not exist, it will be created.
radarTimeZone	NULL String specifying the radar time zone. Default is NULL: extract the time zone from the site table of the 'SQL' database.
targetTimeZone	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".
timeInterval	NULL An optional vector of timestamps (either as `Date` or `POSIXct`) to limit the the data retrieved from the collections table. The filtering is done based on the original radar timezone.
listOfRfFeaturesToExtract	Either NULL (i.e., don't extract any of the rf features), "all" (i.e., extract all rf features) or a vector of the feature numbers to extract. Default is NULL.
siteLocation	Geographic location of the radar measurements in decimal format: c(Latitude, Longitude)
sunOrCivil	optional character variable, Set to "sun" to use sunrise/sunset times or to "civil" to use civil twilight times to group echoes into day/night. Default is "civil".
crepuscule	optional character variable, Set to "nauticalSolar" to use the time between nautical dusk/dawn and sunrise/sunset times to define the crepuscular period, or to "nauticalCivil" to use the time between nautical and civil dusk/dawn to define the crepuscular period, or to "civilSolar" to use the time between civil dusk/dawn and sunrise/sunset times to define the crepuscular period. Default is "nauticalSolar".

Value

a list of R objects with data extracted from the Database: 'echoData', 'protocolData', 'siteData', 'visibilityData', 'timeBinData', 'rfFeatures', 'availableClasses', 'availableBatClasses', 'classProbabilitiesAndMtrFactors', 'batProbabilitiesAndMtrFactors'

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server, database, and other input settings
# ================================================================================
dbDriverChar = "SQL Server" # Set either "SQL Server" or "PostgreSQL"
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
mainOutputDir = file.path(".", "results")
radarTimeZone = "Etc/GMT0"
targetTimeZone = "Etc/GMT0"
listOfRfFeaturesToExtract = c(167, 168)
siteLocation = c(47.494427, 8.716432)
sunOrCivil = "civil"

# Get data
# ================================================================================
dbData = extractDbData(
  dbDriverChar = dbDriverChar,
  dbServer = dbServer,
  dbName = dbName,
  saveDbToFile = TRUE,
  dbDataDir = mainOutputDir,
  radarTimeZone = radarTimeZone,
  targetTimeZone = targetTimeZone,
  listOfRfFeaturesToExtract = listOfRfFeaturesToExtract,
  siteLocation = siteLocation,
  sunOrCivil = sunOrCivil,
  crepuscule = "nauticalSolar"
)

## End(Not run)
```

---

filterData *filterData*

Description

With filterData() both the echo and protocol data can be filtered based on several parameters. The function returns the filtered echo and protocol data.

Usage

```
filterData(
  echoData = NULL,
  protocolData = NULL,
  pulseTypeSelection = NULL,
  rotationSelection = NULL,
  timeRangeTargetTZ = NULL,
  targetTimeZone = "Etc/GMT0",
  classSelection = NULL,
  classProbCutOff = NULL,
  altitudeRange_AGL = NULL,
  manualBlindTimes = NULL,
  echoValidator = FALSE
)
```

Arguments

echoData	data.frame with the echo data from the data list created with extractDbData().
protocolData	data.frame with the protocol data from the data list created with extractDbData() or a subset of it created by filterProtocolData(). Echoes not detected during the listed protocols will be excluded.
pulseTypeSelection	character vector with the pulse types which should be included in the subset. Options: “S”, “M”, “L” (short-, medium-, long-pulse). Default is NULL: no filtering applied based on pulseType.
rotationSelection	numeric vector to select the operation modes with and/or without antenna rotation. Options: 0, 1. (0 = no rotation, 1 = rotation). Default is NULL: no filtering applied based on rotation mode.
timeRangeTargetTZ	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M". Echoes outside the time range will be excluded.
targetTimeZone	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".
classSelection	character string vector with the classes that should be included.
classProbCutOff	numeric cutoff value for class probabilities. Echoes with a lower class probability will be excluded.
altitudeRange_AGL	numeric vector of length 2 with start and end of the altitude range. Echoes outside the altitude range will be excluded.
manualBlindTimes	dataframe with the manual blind times created by the function loadManualBlindTimes().
echoValidator	logical, if set to TRUE, echoes labelled by the echo validator as “non-bio scatterer” will be excluded. If set to FALSE, all echoes are included.

Value

returns the filtered echo and protocol data in the same format as provided in the parameters `echoData` and `protocolData`.

Author(s)

Birgen Haest

See Also

Other filter functions: `filterEchoData()`, `filterProtocolData()`

Examples

```
# Load example data
# ================================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Set input settings for filtering of the data
# ================================================================================
pulseLengthSelection = "S"
rotationSelection = 1
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")
targetTimeZone = "Etc/GMT0"
classSelection = c(
  "passerine_type", "wader_type", "swift_type",
  "large_bird", "unid_bird", "bird_flock"
)
classProbCutoff = NULL
altitudeRange = c(50, 1000)
cManualBlindTimes = NULL
useEchoValidator = FALSE

# Filter the data
# ================================================================================
filteredData = filterData(
  echoData           = dbData$echoData,
  protocolData       = dbData$protocolData,
  pulseTypeSelection = pulseLengthSelection,
  rotationSelection  = rotationSelection,
  timeRangeTargetTZ  = timeRangeData,
  targetTimeZone     = targetTimeZone,
  classSelection     = classSelection,
  classProbCutOff    = classProbCutoff,
  altitudeRange_AGL  = altitudeRange,
  manualBlindTimes   = cManualBlindTimes,
  echoValidator      = useEchoValidator
)
```

filterEchoData *filterEchoData*

Description

With filterEchoData() the echo data can be filtered by several parameters. The function returns the filtered echo data.

Usage

```
filterEchoData(
  echoData = NULL,
  timeRangeTargetTZ = NULL,
  targetTimeZone = "Etc/GMT0",
  protocolData = NULL,
  classSelection = NULL,
  classProbCutOff = NULL,
  altitudeRange_AGL = NULL,
  manualBlindTimes = NULL,
  echoValidator = FALSE
)
```

Arguments

echoData	data.frame with the echo data from the data list created by extractDbData().
timeRangeTargetTZ	Character vector of length 2, with start and end of time range, formatted as "%Y-%m-%d %H:%M". Echoes outside the time range will be excluded.
targetTimeZone	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".
protocolData	data.frame with the protocol data from the data list created by extractDbData() or a subset of it created by filterProtocolData(). Echoes not detected during the listed protocols will be excluded.
classSelection	character string vector with the classes that should be included.
classProbCutOff	numeric cutoff value for class probabilities. Echoes with a lower class probability will be excluded.
altitudeRange_AGL	numeric vector of length 2 with start and end of the altitude range. Echoes outside the altitude range will be excluded.
manualBlindTimes	dataframe with the manual blind times created by the function loadManualBlindTimes().
echoValidator	logical, if set to TRUE, echoes labelled by the echo validator as "non-bio scatterer" will be excluded. If set to FALSE, all echoes are included.

Value

returns the filtered echo data in the same format as provided in the parameter `echoData`.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other filter functions: `filterData()`, `filterProtocolData()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Set input settings for filtering of the data
# ======================================================================
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")
targetTimeZone = "Etc/GMT0"
classSelection = c(
  "passerine_type", "wader_type", "swift_type",
  "large_bird", "unid_bird", "bird_flock"
)
classProbCutoff = NULL
altitudeRange = c(50, 1000)
cManualBlindTimes = NULL
useEchoValidator = FALSE

# Filter the echo data
# ======================================================================
filteredEchoData = filterEchoData(
  echoData          = dbData$echoData,
  timeRangeTargetTZ = timeRangeData,
  targetTimeZone    = targetTimeZone,
  protocolData      = dbData$protocolData,
  classSelection    = classSelection,
  classProbCutOff   = classProbCutoff,
  altitudeRange_AGL = altitudeRange,
  manualBlindTimes  = cManualBlindTimes,
  echoValidator     = useEchoValidator
)
```

filterProtocolData *filterProtocolData*

Description

With filterProtocolData() the protocol data can be filtered by the operation mode (pulse-type and antenna rotation). The function returns the filtered subset of the protocol data which can later be used to filter the echoes based on the operation mode/protocol

Usage

```
filterProtocolData(
  protocolData = NULL,
  pulseTypeSelection = NULL,
  rotationSelection = NULL
)
```

Arguments

protocolData	data.frame with the protocol data from the data list created by extractDbData()
pulseTypeSelection	character vector with the pulse types which should be included in the subset. Options: "S", "M", "L" (short-, medium-, long-pulse). Default is NULL: no filtering applied based on pulseType.
rotationSelection	numeric vector to select the operation modes with and/or without antenna rotation. Options: 0, 1. (0 = no rotation, 1 = rotation). Default is NULL: no filtering applied based on rotation mode.

Value

returns the filtered protocol data in the same format as provided in the parameter protocolData.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other filter functions: filterData(), filterEchoData()

Examples

```
# Load example data
# ========================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
```

```
))

# Set input settings for filtering of the data
# ======================================================================
pulseLengthSelection = "S"
rotationSelection = 1

# Filter the protocol data
# ======================================================================
filteredProtocolData = filterProtocolData(
  protocolData = dbData$protocolData,
  pulseTypeSelection = pulseLengthSelection,
  rotationSelection = rotationSelection
)
```

`filterSpeedFeature37` *Filter outliers in Speed feature (collection.feature37)*

Description

Filter outliers in Speed feature (collection.feature37)

Usage

```
filterSpeedFeature37(echoData = NULL, minEchoDuration = 5)
```

Arguments

`echoData`	valid echodata
`minEchoDuration`	minimum duration of a echo to allow speed feature

Value

echoData with the filtered speed feature 37

Author(s)

Fabian Hertner, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

See Also

Other manipulation functions: `addDayNightInfoPerEcho()`, `computeDensity()`, `computeMTR()`, `convertTimeZone()`, `createVPTS()`, `mergeVisibilityAndManualBlindTimes()`, `reclassToBats()`, `twilight()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Filter speed feature 37
# ======================================================================
minEchoDuration = 5
dbData$echoData = filterSpeedFeature37(
  echoData        = dbData$echoData,
  minEchoDuration = minEchoDuration
)
```

`getBatClassification` *Get a BirdScan 'batClassification' table*

Description

Gets the 'rfClasses' table from a 'Birdscan MR1' 'SQL' database

Usage

```
getBatClassification(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A list containing three variables: (1) batClassificationTable: The 'batClassification' database table; (2) classProbabilitiesAndMtrFactors: A dataframe containing the classification probabilities for all classes for each object; and (3) availableClasses: the classes used for the classification of the objects.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: `QUERY()`, `dbConnectBirdscanSQL()`, `extractDbData()`, `getCollectionTable()`, `getEchoFeatures()`, `getEchoValidationTable()`, `getProtocolTable()`, `getRadarTable()`, `getRfClassification()`, `getSiteTable()`, `getTimeBinsTable()`, `getVisibilityTable()`

Examples

```
## Not run:
# Set server and database settings
# ================================================================================
# Using and Microsoft SQL database
# ================================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ================================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ================================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

batClassification = getBatClassification(dbConnection)

## End(Not run)
```

`getCollectionTable` *Get BirdScan collection table*

Description

Load collection from 'Birdscan MR1' 'SQL' database.

Usage

```
getCollectionTable(dbConnection, dbDriverChar, timeInterval = NULL)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.
`timeInterval`	Null An optional vector of timestamps (either as `Date` or `POSIXct`) to limit the data retrieved from the collections table. The filtering is done based on the original radar timezone.

Value

A dataframe with the collection table

Author(s)

Fabian Hertner, Birgen Haest, Bart Kranstauber

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ================================================================================
# Using and Microsoft SQL database
# ================================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ================================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ================================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

collectionTable = getCollectionTable(dbConnection)

## End(Not run)
```

---

getEchoFeatures *Get BirdScan echo features*

Description

Load echo rffeature map from 'Birdscan MR1' 'SQL' database.

Usage

```
getEchoFeatures(
  dbConnection,
  dbDriverChar,
  listOfRfFeaturesToExtract,
  echoIDRange = NULL
)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.
listOfRfFeaturesToExtract	Either NULL (i.e., don't extract any of the rf features), "all" (i.e., extract all rf features) or a vector of the feature numbers to extract. Default is NULL. Feature IDs can be found in the 'rfFeatures' table in the sql database.
echoIDRange	NULL A two-element vector of integers to subset the rf feature extraction to a range of echoIDs. Default is to extract for all echoes.

Value

A list of the features extracted

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ============================================================================
# Using and Microsoft SQL database
# ============================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ============================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"
```

```
# Open the connection with the database
# ============================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

# Set list of Rf features you also want to extract
# Vector with RF features to extract. Feature IDs can be found in the
# 'rfFeatures' table in the sql database.
# Example: Get wing beat frequency and credibility: c(167, 168)
# Set to NULL to not extract any.
# Set to "all" to extract all features.
# ============================================================================
listOfRfFeaturesToExtract = c(167, 168)

echoFeatures = getEchoFeatures(
  dbConnection,
  listOfRfFeaturesToExtract
)

## End(Not run)
```

`getEchoValidationTable` *Get a BirdScan echo validation table*

Description

Gets the echoValidationTable from an already connected database.

Usage

```
getEchoValidationTable(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A dataframe called echovalidationTable

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
# Using and Microsoft SQL database
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ========================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ========================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

echovalidationTable = getEchoValidationTable(dbConnection)

## End(Not run)
```

---

getManualVisibilityTable

Get manual visibility table

Description

Load visibility table from an already connected 'Birdscan MR1' 'SQL' database.

Usage

```
getManualVisibilityTable(dbConnection, dbDriverChar)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A data.frame with the manual visibility table.

Author(s)

Baptiste Schmid, Birgen Haest

Examples

```
## Not run:
# Set server and database settings
# ================================================================================
# Using and Microsoft SQL database
# ================================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ================================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ================================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

manualVisibilityTable = getManualVisibilityTable(dbConnection)

## End(Not run)
```

---

getProtocolTable *Get BirdScan protocol table*

Description

Load protocol table from an already connected 'Birdscan MR1' 'SQL' database.

Usage

```
getProtocolTable(dbConnection, dbDriverChar)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A dataframe with the protocol table

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
# Using and Microsoft SQL database
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ========================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ========================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

protocolTable = getProtocolTable(dbConnection)

## End(Not run)
```

getRadarTable *Get a BirdScan radar table*

Description

Get the Radar table from an already connected DB and rename the columns appropriately.

Usage

```
getRadarTable(dbConnection, dbDriverChar)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

the radar table as a data frame

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
# Using and Microsoft SQL database
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ========================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"
```

```
# Open the connection with the database
# ==============================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

radarTable = getRadarTable(dbConnection)

## End(Not run)
```

`getRfClassification` *Get a BirdScan 'rfClassification' table*

Description

Gets the 'rfClasses' table from a 'Birdscan MR1' 'SQL' database.

Usage

```
getRfClassification(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A list containing three variables: (1) rfclassificationTable: The 'rfClassification' database table; (2) classProbabilitiesAndMtrFactors: A dataframe containing the classification probabilities for all classes for each object; and (3) availableClasses: the classes used for the classification of the objects.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: `QUERY()`, `dbConnectBirdscanSQL()`, `extractDbData()`, `getBatClassification()`, `getCollectionTable()`, `getEchoFeatures()`, `getEchoValidationTable()`, `getProtocolTable()`, `getRadarTable()`, `getSiteTable()`, `getTimeBinsTable()`, `getVisibilityTable()`

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
# Using and Microsoft SQL database
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ========================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ========================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

rfClassification = getRfClassification(dbConnection)

## End(Not run)
```

`getSiteTable` *Get BirdScan site table*

Description

Load site table from an already connected 'Birdscan MR1' 'SQL' database.

Usage

```
getSiteTable(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A `data.frame` with the site table

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ============================================================================
# Using and Microsoft SQL database
# ============================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ============================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ============================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

siteTable <- getSiteTable(dbConnection)

## End(Not run)
```

---

getTimeBinsTable *Get BirdScan time bins table*

Description

Load time bins table from an already connected 'Birdscan MR1' 'SQL' database.

Usage

```
getTimeBinsTable(dbConnection, dbDriverChar)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A dataframe with the time bins table

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: QUERY(), dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ================================================================================
# Using and Microsoft SQL database
# ================================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ================================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
# ================================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

timeBinsTable = getTimeBinsTable(dbConnection)

## End(Not run)
```

`getVisibilityTable` *Get BirdScan visibility table*

Description

Load visibility table from an already connected 'Birdscan MR1' 'SQL' database.

Usage

```
getVisibilityTable(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	A valid database connection.
`dbDriverChar`	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.

Value

A `data.frame` with the visibility table.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: `QUERY()`, `dbConnectBirdscanSQL()`, `extractDbData()`, `getBatClassification()`, `getCollectionTable()`, `getEchoFeatures()`, `getEchoValidationTable()`, `getProtocolTable()`, `getRadarTable()`, `getRfClassification()`, `getSiteTable()`, `getTimeBinsTable()`

Examples

```
## Not run:
# Set server and database settings
# ================================================================================
# Using and Microsoft SQL database
# ================================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set to "SQL Server"

# Using a PostgreSQL
# ================================================================================
dbServer = "cloud.birdradar.com" # Set the name or IP of your postgreSQL
dbName = "db_Name" # Set the name of your database
dbDriverChar = "PostgreSQL" # Set to "PostgreSQL"

# Open the connection with the database
```

```
# ========================================================================
dbConnection = dbConnectBirdscanSQL(
  dbDriverChar = dbDriverChar,
  dbServer     = dbServer,
  dbName       = dbName,
)

visibilityTable = getVisibilityTable(dbConnection)

## End(Not run)
```

`loadManualBlindTimes` *loadManualBlindTimes*

Description

Load manual blind times from csv file. For the MTR computation the times when the radar was blind have to be known. The radar itself can be blind in case of a protocol change (block time at the beginning of each protocol, usually 60s) or due to rain/snow or clutter (nearby objects, leaves or similar on radome, etc.). These times are stored in the visibility table or in the time_bins table in relation to the time bins duration (5min). To be flexible and not fixed to the 5 min time bins created by the radar, the visibility table is used in this script. In addition to the radar blind times, manual blind times can be defined. Manual blind times have to be defined in a csv file and are loaded with `loadManualBlindTimes()`. An example dataset is available by running: `data(manualBlindTimes) write.csv(manualBlindTimes, file = 'the output file destination', row.names = F)` The file path is defined as a global variable 'manualBlindTimesFile'. A custom file and filepath can be used instead. The manual blind times have to be entered with 3 columns: start time 'yyyy-mm-dd hh:mm:ss', stop time 'yyyy-MM-dd hh:mm:ss', type.

Example: 2021-01-16 04:15:00,2021-01-16 05:42:00,rain 2021-01-17 16:33:00,2021-01-17 18:04:00,clutter Manual blind time types can be chosen freely. When computing observation times, it can be decided if some of the defined manual blind time types should be treated as observed time with MTR zero or as blind time (e.g. rain). If no file is present or the file is empty, no manual blind times will be computed.

Usage

```
loadManualBlindTimes(filePath, blindTimesTZ, targetTZ)
```

Arguments

`filePath`	character string, absolute filepath of the manual blind time file
`blindTimesTZ`	time zone of the blind times
`targetTZ`	target time zone of the blind times

Value

A dataframe with the manual blind times

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other sample data: CH_Sempach_2024_SEP24_25_DataExtract, classAbbreviations, manualBlindTimes

Examples

```
# Load manual blind time example data from birdscanR package
# ============================================================================
data(manualBlindTimes)

# Save example manual blind times to a temporary file
# ============================================================================
tmpFile = tempfile(fileext = ".csv")
write.table(manualBlindTimes,
  file = tmpFile, sep = ",",
  row.names = FALSE, col.names = FALSE
)

# Read the manual blind times from file
# ============================================================================
manualBlindTimes.new = loadManualBlindTimes(
  filePath     = tmpFile,
  blindTimesTZ = "Etc/GMT0",
  targetTZ     = "Etc/GMT0"
)
```

`manualBlindTimes` *Example file on how to include manual blind times for your 'Birdscan MR1' database.*

Description

To create your own manual blind times file, just copy this file, and adjust.

Usage

```
data(manualBlindTimes)
```

Format

An object of class `data.frame`.

See Also

Other sample data: CH_Sempach_2024_SEP24_25_DataExtract, classAbbreviations, loadManualBlindTimes()

Examples

```
data(manualBlindTimes)
```

`mergeVisibilityAndManualBlindTimes` *mergeVisibilityAndManualBlindTimes*

Description

Function to merge manual blind times with blind times from visibility table. For further processing the radar (visibility) and manual blind times have to be merged with `mergeVisibilityAndManualBlindTimes()`. This function will add a blind time type to the radar/visibility blind times. Blind times during the block time (usually 60s) at the beginning of each protocol are given the type 'protocolChange', the rest of the radar blind times are given the type "visibility". After that the visibility and manual blind times will be merged. In case manual blind times and radar blind times are overlapping, radar blind times with type "visibility" will be overwritten, but not radar blind times with type "protocolChange".

Usage

```
mergeVisibilityAndManualBlindTimes(
  visibilityData,
  manualBlindTimes = NULL,
  protocolData
)
```

Arguments

`visibilityData`	dataframe with the visibility data from the data list created by `extractDbData()`.
`manualBlindTimes`	dataframe with the manual blind times created by the function 'loadManualBlindTimes'.
`protocolData`	dataframe with the protocol data from the data list created by `extractDbData()` or a subset of it created by the function `filterProtocolData()`.

Value

dataframe with overall blind times

Author(s)

Fabian Hertner, Birgen Haest, Baptiste Schmid

See Also

Other manipulation functions: `addDayNightInfoPerEcho()`, `computeDensity()`, `computeMTR()`, `convertTimeZone()`, `createVPTS()`, `filterSpeedFeature37()`, `reclassToBats()`, `twilight()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Get manual blind times
# ======================================================================
data(manualBlindTimes)
tmpFile = tempfile(fileext = ".csv")
write.table(manualBlindTimes,
  file = tmpFile, sep = ",",
  row.names = FALSE, col.names = FALSE
)
cManualBlindTimes = loadManualBlindTimes(
  filePath     = tmpFile,
  blindTimesTZ = "Etc/GMT0",
  targetTZ     = "Etc/GMT0"
)

# Merge manual and automatic blind times
# ======================================================================
blindTimes = mergeVisibilityAndManualBlindTimes(
  visibilityData   = dbData$visibilityData,
  manualBlindTimes = cManualBlindTimes,
  protocolData     = dbData$protocolData
)
```

`plotExploration` *plotExploration*

Description

This function creates a time series plot showing all of the observed echoes at their respective altitudes. These plots are helpful to roughly visually explore your data (and for example spot oddities).

Usage

```
plotExploration(
  echoData = NULL,
  timeRange = NULL,
  targetTimeZone = "Etc/GMT0",
  manualBlindTimes = NULL,
  visibilityData = NULL,
  protocolData = NULL,
```

```
  sunriseSunset = NULL,
  maxAltitude = NULL,
  filePath = NULL
)
```

Arguments

echoData	dataframe with the echo data from the data list created by `extractDbData()` or a subset of it created by the function 'filterEchoData'
timeRange	optional list of string vectors length 2, start and end time of the time ranges that should be plotted. The date/time format is "yyyy-MM-dd hh:mm". If not set, all echo data is plotted in one plot. Note: Too long time-ranges may produce an error if the created image is too large and the function can't allocate the file.
targetTimeZone	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".
manualBlindTimes	optional dataframe with the manual blind times created by `loadManualBlindTimes()`. If not set, manual blind times are not shown in the plot.
visibilityData	optional dataframe with the visibility data created by `extractDbData()`. If not set, visibility data are not shown in the plot.
protocolData	optional dataframe with the protocol data used to filter the echoes, created by `extractDbData()` or a subset of it created by `filterProtocolData()`. If not set, periods without a protocol are not shown in the plot.
sunriseSunset	optional dataframe with sunrise/sunset, civil, and nautical twilight times created by `twilight()`. If not set, day/night times are not shown in the plot.
maxAltitude	optional numeric, fixes the maximum value of the y-Scale of the plot to the given value. If negative or not set, the y-Scale is auto-scaled.
filePath	character string, path of the directory where the plot should be saved. `savePlotToFile()` is used to save the plots as png files with an auto-generated filename.

Value

png files stored in the directory specified in 'filePath'

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other plot functions: `createTimeRangeForPlot()`, `plotLongitudinalMTR()`

Examples

```
# Load example data
# ================================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
```

```
))

# Set manual blind times to NULL (no manual blind times)
# ============================================================================
cManualBlindTimes = NULL

# Make Plot
# ============================================================================
timeRangePlot = list(
  c("2024-09-24 00:00", "2024-09-24 23:59"),
  c("2024-09-25 00:00", "2024-09-25 23:59")
)
plotExploration(
  echoData         = dbData$echoData,
  timeRange        = timeRangePlot,
  targetTimeZone   = "Etc/GMT0",
  manualBlindTimes = cManualBlindTimes,
  visibilityData   = dbData$visibilityData,
  protocolData     = dbData$protocolData,
  sunriseSunset    = dbData$sunriseSunset,
  maxAltitude      = -1,
  filePath         = tempdir()
)
```

---

plotLongitudinalMTR *plotLongitudinalMTR*

Description

Plots a time series of MTR values as a bar plot. For each bar the spread (first and third Quartile) is shown as error bars as well as the numbers of echoes. Periods with no observation are indicated with grey, negative bars.

Usage

```
plotLongitudinalMTR(
  mtr,
  maxMTR,
  timeRange = NULL,
  targetTimeZone = "Etc/GMT0",
  plotClass = "allClasses",
  propObsTimeCutoff = 0.2,
  plotSpread = TRUE,
  filePath = NULL
)
```

Arguments

mtr	data frame with MTR values created by computeMTR().
maxMTR	optional numeric variable, fixes the maximum value of the y-Scale of the plot to the given value. If negative or not set, the y-Scale is auto-scaled.
timeRange	optional list of string vectors length 2, start and end time of the time ranges that should be plotted. The date/time format is "yyyy-MM-dd hh:mm".
targetTimeZone	"Etc/GMT0" String specifying the target time zone. Default is "Etc/GMT0".
plotClass	character string with the class of which the MTR data should be plotted. If not set or set to "allClasses", MTR of all classes will be plotted.
propObsTimeCutoff	numeric between 0 and 1. If the MTR is computed per day and night, time bins with a proportional observation time smaller than propObsTimeCutoff are ignored when combining the time bins. If the MTR is computed for each time bin, the parameter is ignored.
plotSpread	logical, choose if the spread (first and third quartile) should be plotted.
filePath	character string, path of the directory where the plot should be saved. savePlotToFile() is used to save the plots as png files with an auto-generated filename.

Value

png files stored in the directory specified with 'filePath'

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other plot functions: createTimeRangeForPlot(), plotExploration()

Examples

```
# Load example data
# ================================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
dbName = "CH_Sempach_2024_SEP24_25"
targetTimeZone = "Etc/GMT0"
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")

# Set manual blind times to NULL (no manual blind times)
# ================================================================================
cManualBlindTimes = NULL

# Compute migration traffic rate
# ================================================================================
```

```
classSelection.mtr = c("passerine_type")
mtrData = computeMTR(
  dbName                       = dbName,
  echoes                       = dbData$echoData,
  classSelection               = classSelection.mtr,
  altitudeRange                = c(25, 1025),
  altitudeBinSize              = 50,
  timeRange                    = timeRangeData,
  timeBinDuration_sec          = 1800,
  timeZone                     = targetTimeZone,
  sunriseSunset                = dbData$sunriseSunset,
  sunOrCivil                   = "civil",
  protocolData                 = dbData$protocolData,
  visibilityData               = dbData$visibilityData,
  manualBlindTimes             = cManualBlindTimes,
  saveBlindTimes               = FALSE,
  blindTimesOutputDir          = getwd(),
  blindTimeAsMtrZero           = NULL,
  propObsTimeCutoff            = 0,
  computePerDayNight           = FALSE,
  computeAltitudeDistribution  = TRUE
)

# Make Plot
# ============================================================================
timeRangePlot = list(
  c("2024-09-24 00:00", "2024-09-24 23:59"),
  c("2024-09-25 00:00", "2024-09-25 23:59")
)
plotLongitudinalMTR(
  mtr               = mtrData,
  maxMTR            = -1,
  timeRange         = timeRangePlot,
  targetTimeZone    = "Etc/GMT0",
  plotClass         = "allClasses",
  propObsTimeCutoff = 0.2,
  plotSpread        = TRUE,
  filePath          = tempdir()
)
```

QUERY *Query 'SQL' database*

Description

Run an 'SQL' query on an already connected database.

Usage

```
QUERY(dbConnection, dbDriverChar, query, as.is = FALSE)
```

Arguments

dbConnection	A valid database connection.
dbDriverChar	This was the name of the driver, and is now automatically detected, therefore it should be omitted into the future.
query	an 'SQL' string with your query
as.is	If TRUE, leaves data as it is

Value

the result of the query

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other read SQL database functions: dbConnectBirdscanSQL(), extractDbData(), getBatClassification(), getCollectionTable(), getEchoFeatures(), getEchoValidationTable(), getProtocolTable(), getRadarTable(), getRfClassification(), getSiteTable(), getTimeBinsTable(), getVisibilityTable()

Examples

```
## Not run:
# Set server and database settings
# ========================================================================
dbServer = "MACHINE\\SERVERNAME" # Set the name of your SQL server
dbName = "db_Name" # Set the name of your database
dbDriverChar = "SQL Server" # Set either "SQL Server" or "PostgreSQL"

# Open the connection with the database
# ========================================================================
dsn = paste0(
  "driver=", dbDriverChar, ";server=", dbServer,
  ";database=", dbName,
  ";uid=", rstudioapi::askForPassword("Database user"),
  ";pwd=", rstudioapi::askForPassword("Database password")
)
dbConnection = RODBC::odbcDriverConnect(dsn)

QUERY(
  dbConnection = dbConnection,
  query = "Select * From collection order by row asc"
)

## End(Not run)
```

`reclassToBats` *integrate bat classification*

Description

Reclassifies echoes based on bat classification.

Usage

```
reclassToBats(
  echoData = NULL,
  batProbabilitiesAndMtrFactors = NULL,
  reclassToBatCutoff = -1
)
```

Arguments

`echoData`	echodata dataframe, output from extractDbData
`batProbabilitiesAndMtrFactors`	probabilities of bat classification, output from extractDbData'
`reclassToBatCutoff`	Threshold (0..1), classification of echoes with bat probability higher than reclassToBatCutoff will be set to 'bat'

Value

echoData dataframe

Author(s)

Fabian Hertner

See Also

Other manipulation functions: `addDayNightInfoPerEcho()`, `computeDensity()`, `computeMTR()`, `convertTimeZone()`, `createVPTS()`, `filterSpeedFeature37()`, `mergeVisibilityAndManualBlindTimes()`, `twilight()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))

# Reclass To Bats
```

```
# ========================================================================
dbData$echoData = reclassToBats(
  echoData = dbData$echoData,
  batProbabilitiesAndMtrFactors = dbData$batProbabilitiesAndMtrFactors,
  reclassToBatCutoff = 0.5
)
```

---

saveMTR *saveMTR*

Description

Saves MTR data to a .rds file in the directory `filepath`. If the directory is not existing it will be created if possible.

Usage

```
saveMTR(
  mtr,
  filepath,
  fileName = NULL,
  fileNamePrefix = NULL,
  dbName = NULL,
  rotSelection = NULL,
  pulseTypeSelection = NULL,
  classAbbreviations = NULL
)
```

Arguments

`mtr`	dataframe with MTR values created by `computeMTR()`
`filepath`	character string, path of the directory. If the directory does not exist it will be created if possible.
`fileName`	Filename (string) for the file. If not set, the filename will be built using the input of the variables 'filenamePrefix', 'dbName', 'classAbbreviations', and other info in the 'mtr' data. If set, overrides the automatic filename creation.
`fileNamePrefix`	prefix of the filename (string). If not set, "mtr" is used. Different information about the MTR data will be appended to the filename.
`dbName`	character string, name of the database. Used to create the filename, if 'fileName' is not provided.
`rotSelection`	numeric vector, rotation selection which was used to filter protocols. Used to create the filename, if 'fileName' is not provided. If not set, the rotation selection will not be appended to the filename.

`pulseTypeSelection`
: character vector, pulse type selection which was used to filter protocols. Used to create the filename, if 'fileName' is not provided. If not set, the pulse type selection will not be appended to the filename.

`classAbbreviations`
: Two-column dataframe with character first column named 'class' and character second 'abbr', containing the full names of the classes and their abbreviations to use in the output filename. Default = NULL, meaning the abbreviations will be used that are stored in the package; See data(classAbbreviations). Used to create the filename, if 'fileName' is not provided.

Value

No return value, used to save MTR to file.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other write file functions: `compileData()`, `savePlotToFile()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
dbName = "CH_Sempach_2024_SEP24_25"
mainOutputDir = tempdir()
targetTimeZone = "Etc/GMT0"
timeRangeData = c("2024-09-24 00:00", "2024-09-25 23:59")

# Set manual blind times to NULL (no manual blind times)
# ======================================================================
cManualBlindTimes = NULL

# Compute migration traffic rate
# ======================================================================
classSelection.mtr = c("passerine_type")
mtrData = computeMTR(
  dbName                      = dbName,
  echoes                      = dbData$echoData,
  classSelection              = classSelection.mtr,
  altitudeRange               = c(25, 1025),
  altitudeBinSize             = 50,
  timeRange                   = timeRangeData,
  timeBinDuration_sec         = 1800,
  timeZone                    = targetTimeZone,
```

```
  sunriseSunset               = dbData$sunriseSunset,
  sunOrCivil                  = "civil",
  protocolData                = dbData$protocolData,
  visibilityData              = dbData$visibilityData,
  manualBlindTimes            = cManualBlindTimes,
  saveBlindTimes              = FALSE,
  blindTimesOutputDir         = mainOutputDir,
  blindTimeAsMtrZero          = NULL,
  propObsTimeCutoff           = 0,
  computePerDayNight          = FALSE,
  computeAltitudeDistribution = TRUE
)

# Save the MTR data to file
# ============================================================================
saveMTR(
  mtr      = mtrData,
  filepath = mainOutputDir,
  dbName   = dbName
)
```

---

savePlotToFile *savePlotToFile*

Description

Saves created plots as .png.

Usage

```
savePlotToFile(
  plot = NULL,
  filePath = NULL,
  plotType = NULL,
  plotWidth_mm = NULL,
  plotHeight_mm = NULL,
  timeRange = NULL,
  classSelection = NULL,
  altitudeRange = NULL,
  classAbbreviations = NULL
)
```

Arguments

plot	'ggplot' plot to be saved
filePath	character string, path of the directory, e.g. "your-project-directory/Data/MTR". If the directory does not exist it will be created if possible.

plotType	character string, name/description of the plot, used to create the filename. If not set, the pulse type selection will not be appended to the filename
plotWidth_mm	numeric, width of the plot in mm. If not set, the size of the png will be set automatically.
plotHeight_mm	numeric, height of the plot in mm. If not set, the size of the png will be set automatically.
timeRange	POSIXct vector of size 2, time range of the plot, used to create the filename. If not set, the pulse type selection will not be appended to the filename
classSelection	character string vector, classes that were used to create the plot, used to create the filename. If not set, the pulse type selection will not be appended to the filename
altitudeRange	numeric vector of size 2, altitude range used to create the plot, used to create the filename. If not set, the pulse type selection will not be appended to the filename
classAbbreviations	Two-column dataframe with character first column named 'class' and character second 'abbr', containing the full names of the classes and their abbreviations to use in the output filename. Default = NULL, meaning the abbreviations will be used that are stored in the package; See data(classAbbreviations).

Value

No return value, used to save plots to file.

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other write file functions: `compileData()`, `saveMTR()`

Examples

```
# Load example data
# ======================================================================
dbData = readRDS(system.file("extdata",
  "CH_Sempach_2024_SEP24_25_DataExtract.rds",
  package = "birdscanR"
))
mainOutputDir = tempdir()

# Set manual blind times to NULL (no manual blind times)
# ======================================================================
cManualBlindTimes = NULL

# Make Plot
# ======================================================================
timeRangePlot = list(
  c("2024-09-24 00:00", "2024-09-24 23:59"),
```

```
  c("2024-09-25 00:00", "2024-09-25 23:59")
)
cPlot = plotExploration(
  echoData         = dbData$echoData,
  timeRange        = timeRangePlot,
  targetTimeZone   = "Etc/GMT0",
  manualBlindTimes = cManualBlindTimes,
  visibilityData   = dbData$visibilityData,
  protocolData     = dbData$protocolData,
  sunriseSunset    = dbData$sunriseSunset,
  maxAltitude      = -1,
  filePath         = mainOutputDir
)

# Save plot
# ============================================================================
savePlotToFile(
  plot          = cPlot,
  filePath      = mainOutputDir,
  plotType      = "S",
  plotWidth_mm  = 400,
  plotHeight_mm = 200
)
```

`twilight` *Get the nautical, civil, and solar dawn and dusk for a given timerange and locations.*

Description

Get the time of nautical (sun at 12 degrees below horizon), civil (sun at 6 degrees below horizon) and solar (sun at 0 degrees below horizon) dawn and dusk for each day over a given time range.

Usage

```
twilight(timeRange, latLon, crs_datum = "WGS84", timeZone)
```

Arguments

`timeRange`	A two-element character vector with elements of the form %Y-%m-%d defining the start and end of the timerange for which you want to get the twilight information.
`latLon`	A list of X, Y coordinates
`crs_datum`	The coordinate reference system and datum of the X, Y coordinates. Default = "WGS84.
`timeZone`	The time zone of the area of interest

Value

A data frame with the results

Author(s)

Fabian Hertner, Birgen Haest

See Also

Other manipulation functions: addDayNightInfoPerEcho(), computeDensity(), computeMTR(), convertTimeZone(), createVPTS(), filterSpeedFeature37(), mergeVisibilityAndManualBlindTimes(), reclassToBats()

Examples

```
sunriseSunset = twilight(
  timeRange = c("2024-09-24 00:00", "2024-09-25 23:59"),
  latLon    = c(47.12764, 8.192569),
  timeZone  = "Etc/GMT0"
)
```

Index