Package 'birdscanR'

July 22, 2025

Version 0.3.0

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

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>.

Type Package

License GPL-3

Language en-gb

Encoding UTF-8

RoxygenNote 7.3.2

VignetteBuilder knitr

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

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

LazyData true

Depends R (>= 3.5.0)

Imports DBI, dplyr, ggplot2, grDevices, magrittr, suntools, methods, modi, reshape2, RODBC, RPostgreSQL, rlang, rstudioapi, sp, stats, tibble, tidyr, utils

Suggests knitr

NeedsCompilation no

Author Birgen Haest [aut, cre] (ORCID:
<https://orcid.org/0000-0002-8739-6460>),
Fabian Hertner [aut],
Baptiste Schmid [ctb],
Damiano Preatoni [ctb],
Johannes De Groeve [ctb],
Felix Liechti [ctb]

Maintainer Birgen Haest <birgen.haest@vogelwarte.ch>

Repository CRAN

Date/Publication 2024-07-05 09:50:06 UTC

Contents

addDayNightInfoPerEcho *addDayNightInfoPerEcho*

Description

The function '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 by the function 'extractDBData'
sunriseSunset	dataframe with sunrise/sunset and civil twilight times created by the function '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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input settings for data extraction
# ======================================================================
  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"
  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")

# Get sunrise/sunset information
# ======================================================================
  sunrisesunset = twilight(timeRange = c("2021-01-15 00:00",
                                         "2021-01-31 00:00"),
                           latLon    = siteLocation,
                           timeZone  = targetTimeZone)

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

## End(Not run)
```

`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".

Examples

```
data(classAbbreviations)
```

---

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 by the function 'extractDBData' or a subset of it created by the function 'filterEchoData'.
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 the function '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 the function `extractDBData` or a subset of it created by the function `filterProtocolData`.
visibilityData	dataframe with the visibility data from the data list created by the function '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. If the density 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 density is computed for each time bin, the parameter is ignored.

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, <birgen.haest@vogelwarte.ch>; Fabian Hertner, <fabian.hertner@swiss-birdradar.com>; Baptiste Schmid, <baptiste.schmid@vogelwarte.ch>;

Examples

```
## Not run:
# Set server, database, and other input 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"
  mainOutputDir  = file.path(".", "results")
  radarTimeZone  = "Etc/GMT0"
  targetTimeZone = "Etc/GMT0"
  listOfRfFeaturesToExtract = c(167, 168)
  siteLocation   = c(47.494427, 8.716432)
  sunOrCivil     = "civil"
  crepuscule     = "nauticalSolar"
  timeRangeData  = c("2021-01-15 00:00", "2021-01-31 00:00")

# 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                    = crepuscule)

# Get sunrise/sunset
# ======================================================================
  sunriseSunset = twilight(timeRange = timeRangeData,
                           latLon    = c(47.494427, 8.716432),
                           timeZone  = targetTimeZone)

# Get manual blind times
# ======================================================================
  data(manualBlindTimes)
  cManualBlindTimes = manualBlindTimes

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

## End(Not run)
```

---

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 the function 'extractDBData' or a subset of it created by the function '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 the function '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 the function `extractDBData` or a subset of it created by the function `filterProtocolData`.

`visibilityData` dataframe with the visibility data from the data list created by the function '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. 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.

`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, <fabian.hertner@swiss-birdradar.com>; Baptiste Schmid, <baptiste.schmid@vogelwarte.ch>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input 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"
  mainOutputDir  = file.path(".", "results")
  radarTimeZone  = "Etc/GMT0"
  targetTimeZone = "Etc/GMT0"
  listOfRfFeaturesToExtract = c(167, 168)
  siteLocation   = c(47.494427, 8.716432)
  sunOrCivil     = "civil"
  crepuscule     = "nauticalSolar"
  timeRangeData  = c("2021-01-15 00:00", "2021-01-31 00:00")

# 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                    = crepuscule)

# Get sunrise/sunset
# ======================================================================
  sunriseSunset = twilight(timeRange = timeRangeData,
                           latLon    = c(47.494427, 8.716432),
                           timeZone  = targetTimeZone)

# Get manual blind times
# ======================================================================
```

```
  data(manualBlindTimes)
  cManualBlindTimes = manualBlindTimes

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

## End(Not run)
```

`computeObservationTime` *computeObservationTime*

Description

Compute blind times and observation times during time bins based on protocol data and blind times

Usage

```
computeObservationTime(
  timeBins,
  protocolData,
  blindTimes,
  blindTimeAsMtrZero = NULL
)
```

Arguments

timeBins	dataframe with the time bins created by the function createTimeBins.
protocolData	dataframe with the protocol data from the data list created by the function extractDBData or a subset of it created by the function filterProtocolData.
blindTimes	dataframe containing the blind times created by the function mergeVisibilityAndManualBlindTimes.
blindTimeAsMtrZero	character string vector with the blind time types which should be treated as observation time with MTR zero.

Value

returns a dataframe with the time bins completed with the observation times of each time bin.

Author(s)

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

---

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

Examples

```
## Not run:
# Set server, database, and other input settings for data extraction
# ======================================================================
  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"
  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)

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

## End(Not run)
```

---

createTimeBins *createTimeBins*

Description

Create time bins with a given duration. Time bins expanding over a day/night change will be split in two time bins.

Usage

```
createTimeBins(
  timeRange,
  timeBinDuration_sec,
  timeZone,
  sunriseSunset,
  dnBins = TRUE,
  crepBins = FALSE,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar"
)
```

Arguments

timeRange	vector of length 2, with start and end of time range as POSIXct
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, civil dawn/dusk. computed with function 'twilight'
dnBins	Logical. Default TRUE. Determines whether timebins based on day/night values (determined by the parameter 'sunOrCivil') are created.
crepBins	Logical. Default FALSE. Determines whether timebins with crepuscular time phases are created (determined by the parameter 'crepuscule').
sunOrCivil	sunrise/sunset or civil dawn/dusk used to split day and night. Supported values: "sun" or "civil", default: "civil"
crepuscule	Used to split into crepusculeMorning, day, crepusculeEvening, and night. 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

returns a dataframe with the time bins information

Author(s)

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

`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,
  saveDbToFile = FALSE,
  dbDataDir = NULL,
  radarTimeZone = NULL,
  targetTimeZone = "Etc/GMT0",
  listOfRfFeaturesToExtract = NULL,
  siteLocation = NULL,
  sunOrCivil = "civil",
  crepuscule = "nauticalSolar"
)
```

Arguments

`dbDriverChar`	'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'. If 'PostgreSQL', it connects to cloud.birdradar.com
`dbServer`	NULL The name of the Server
`dbName`	NULL The name of the Database
`dbUser`	NULL The USER name of the Server
`dbPwd`	NULL The password for the user name
`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".
`listOfRfFeaturesToExtract`	NULL or a list of feature to extract
`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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input 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"
  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 the function filterData both the echo and protocol data can be filtered by 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	dataframe with the echo data from the data list created by the function extractDBData.
protocolData	dataframe with the protocol data from the data list created by the function extractDBData or a subset of it created by the function 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, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input settings for data extraction
# ============================================================================
  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"
  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)

# Set input settings for filtering of the data
# ============================================================================
  pulseLengthSelection = "S"
  rotationSelection    = 1
  timeRangeData        = c("2021-01-15 00:00", "2021-01-31 00:00")
  classSelection       = c("passerine_type", "wader_type", "swift_type",
                           "large_bird", "unid_bird", "bird_flock")
```

```
  classProbCutoff      = NULL
  altitudeRange        = c(50, 1000)
  data(manualBlindTimes)
  cManualBlindTimes    = manualBlindTimes
  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)

## End(Not run)
```

`filterEchoData` *filterEchoData*

Description

With the function `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`	dataframe with the echo data from the data list created by the function `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`
: dataframe with the protocol data from the data list created by the function `extractDBData` or a subset of it created by the function `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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input settings for data extraction
# ======================================================================
  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"
  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)

  # Set input settings for filtering of the data
  # ============================================================================
    timeRangeData     = c("2021-01-15 00:00", "2021-01-31 00:00")
    classSelection    = c("passerine_type", "wader_type", "swift_type",
                          "large_bird", "unid_bird", "bird_flock")
    classProbCutoff   = NULL
    altitudeRange     = c(50, 1000)
    data(manualBlindTimes)
    cManualBlindTimes = manualBlindTimes
    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)

  ## End(Not run)
```

`filterProtocolData` *filterProtocolData*

Description

With the function `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	dataframe with the protocol data from the data list created by the function 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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input settings for data extraction
# ======================================================================
  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"
  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)

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

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

## End(Not run)
```

`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`	dbDriverChar 'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'. If 'PostgreSQL', it connects to cloud.birdradar.com

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

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)

rfClassification = getBatClassification(dbConnection, dbDriverChar)

## End(Not run)
```

`getCollectionTable` *Get BirdScan collection table*

Description

load collection from 'Birdscan MR1' 'SQL' database

Usage

```
getCollectionTable(dbConnection, dbDriverChar)
```

Arguments

`dbConnection`	a valid database connection
`dbDriverChar`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the collection table

Author(s)

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

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)

collectionTable = getCollectionTable(dbConnection, dbDriverChar)

## End(Not run)
```

`getEchoFeatures` *Get BirdScan echo features*

Description

load echo rffeature map from 'Birdscan MR1' 'SQL' database

Usage

```
getEchoFeatures(dbConnection, dbDriverChar, listOfRfFeaturesToExtract)
```

Arguments

`dbConnection`	a valid database connection
`dbDriverChar`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com
`listOfRfFeaturesToExtract`	a list of feature to extract

Value

A list of the features extracted

Author(s)

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

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)
```

```
# 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.
# ============================================================================
  listOfRfFeaturesToExtract = c(167, 168)

echoFeatures = getEchoFeatures(dbConnection, dbDriverChar,
                               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`	dbDriverChar 'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'. If 'PostgreSQL', it connects to cloud.birdradar.com

Value

A dataframe called echovalidationTable

Author(s)

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

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)

echovalidationTable = getEchoValidationTable(dbConnection, dbDriverChar)

## 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`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the manual visibility table

Author(s)

Baptiste Schmid <baptiste.schmid@vogelwarte.ch>; Birgen Haest <birgen.haest@vogelwarte.ch>

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)

manualVisibilityTable = getManualVisibilityTable(dbConnection, dbDriverChar)

## 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`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the protocol table

Author(s)

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

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)

protocolTable = getProtocolTable(dbConnection, dbDriverChar)

## 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`	the name of the driver.

Value

the radar table as a data frame

Author(s)

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

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)
```

```
radarTable = getRadarTable(dbConnection, dbDriverChar)

## 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`	dbDriverChar 'SQL Server' The name of the driver. Should be either 'SQL Server' or 'PostgreSQL'. If 'PostgreSQL', it connects to cloud.birdradar.com

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

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)
```

```
rfClassification = getRfClassification(dbConnection, dbDriverChar)

## 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`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the site table

Author(s)

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

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)

siteTable = getSiteTable(dbConnection, dbDriverChar)

## 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	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the time bins table

Author(s)

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

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)

timeBinsTable = getTimeBinsTable(dbConnection, dbDriverChar)

## 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`	the name of the driver. If different from 'PostgreSQL' it connects to cloud.birdradar.com

Value

A dataframe with the visibility table

Author(s)

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

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)

visibilityTable = getVisibilityTable(dbConnection, dbDriverChar)

## 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 the function 'loadManualBlindTimes'. A 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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# load manual blind time example data from birdscanR package
  data(manualBlindTimes)
```

```
# Save example manual blind times to a file
  write.csv(manualBlindTimes, file = "manualBlindTimes.csv", row.names = F)

# Read the manual blind times from file
  manualBlindTimes.new = loadManualBlindTimes(filePath     = "./manualBlindTimes.csv",
                                              blindTimesTZ = "ETC/GMT",
                                              targetTZ     = "ETC/GMT")

## End(Not run)
```

`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"`.

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 the function '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 the function '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 the function 'extractDBData' or a subset of it created by the function 'filterProtocolData'.

Value

dataframe with overall blind times

Author(s)

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

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)

# Get visibility table
# ================================================================================
  visibilityTable = getVisibilityTable(dbConnection, dbDriverChar)

# Get manual blind times
# ================================================================================
  data(manualBlindTimes)
  cManualBlindTimes = manualBlindTimes
```

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

## End(Not run)
```

---

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 the function '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 the function 'loadManualBlindTimes'. If not set, manual blind times are not shown in the plot.
visibilityData	optional dataframe with the visibility data created by the function '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 the function 'extractDBData' or a subset of it created by the function '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 the function '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. The function '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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
#' # Set server, database, and other input 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"
  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)

# Get manual blindtimes
# ========================================================================
  data("manualBlindTimes")
  cManualBlindTimes = manualBlindTimes
```

```
# Make Plot
# ============================================================================
  timeRangePlot = list(c("2021-01-15 00:00", "2021-01-22 00:00"),
                       c("2021-01-23 00:00", "2021-01-31 00:00"))
  plotExploration(echoData         = dbData$echoData,
                  timeRange        = timeRangePlot,
                  targetTimeZone   = "Etc/GMT0",
                  manualBlindTimes = cManualBlindTimes,
                  visibilityData   = dbData$visibilityData,
                  protocolData     = dbData$protocolData,
                  sunriseSunset    = dbData$sunriseSunset,
                  maxAltitude      = -1,
                  filePath         = "./")

## End(Not run)
```

---

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 the function '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. The function '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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input 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"
  mainOutputDir  = file.path(".", "results")
  radarTimeZone  = "Etc/GMT0"
  targetTimeZone = "Etc/GMT0"
  listOfRfFeaturesToExtract = c(167, 168)
  siteLocation   = c(47.494427, 8.716432)
  sunOrCivil     = "civil"
  timeRangeData       = c("2021-01-15 00:00", "2021-01-31 00:00")

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

# Get sunrise/sunset
# ========================================================================
```

```
  sunriseSunset = twilight(timeRange = timeRangeData,
                           latLon    = c(47.494427, 8.716432),
                           timeZone  = targetTimeZone)

# Get manual blind times
# ============================================================================
  data(manualBlindTimes)
  cManualBlindTimes = manualBlindTimes

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

# Make Plot
# ============================================================================
  timeRangePlot = list(c("2021-01-15 00:00", "2021-01-22 00:00"),
                       c("2021-01-23 00:00", "2021-01-31 00:00"))
  plotExplorationplotLongitudinalMTR(mtr               = mtrData,
                                     maxMTR            = -1,
                                     timeRange         = timeRangePlot,
                                     targetTimeZone    = "Etc/GMT0",
                                     plotClass         = "allClasses",
                                     propObsTimeCutoff = 0.2,
                                     plotSpread        = TRUE,
                                     filePath          = "./")

## End(Not run)
```

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	the name of the driver
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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

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,
        dbDriverChar = dbDriverChar,
        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,
  batClassProbabilitiesAndMtrFactors = NULL,
  reclassToBatCutoff = -1
)
```

Arguments

`echoData`	echodata dataframe, output from extractDbData
`batClassProbabilitiesAndMtrFactors`	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, <fabian.hertner@swiss-birdradar.com>

Examples

```
## Not run:
# Set server, database, and other input 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"
  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")
#'
# Reclass To Bats
# ============================================================================
  dbData$echoData = reclassToBats(echoData = dbData$echoData,
                                  batClassProbabilitiesAndMtrFactors =
                                      dbData$batClassProbabilitiesAndMtrFactors,
                                  reclassToBatCutoff = 0.5)

## End(Not run)
```

---

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 the function `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, <fabian.hertner@swiss-birdradar.com>; Birgen Haest, <birgen.haest@vogelwarte.ch>

Examples

```
## Not run:
# Set server, database, and other input 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"
  mainOutputDir  = file.path(".", "results")
  radarTimeZone  = "Etc/GMT0"
  targetTimeZone = "Etc/GMT0"
  listOfRfFeaturesToExtract = c(167, 168)
  siteLocation   = c(47.494427, 8.716432)
  sunOrCivil     = "civil"
  timeRangeData       = c("2021-01-15 00:00", "2021-01-31 00:00")

# Get data
# ======================================================================
  dbData = extractDbData(dbDriverChar                  = dbDriverChar,
                         dbServer                      = dbServer,
```

```
                          dbName                          = dbName,
                          saveDbToFile                    = TRUE,
                          dbDataDir                       = mainOutputDir,
                          radarTimeZone                   = radarTimeZone,
                          targetTimeZone                  = targetTimeZone,
                          listOfRfFeaturesToExtract       = listOfRfFeaturesToExtract,
                          siteLocation                    = siteLocation,
                          sunOrCivil                      = sunOrCivil)

# Get sunrise/sunset
# ========================================================================
  sunriseSunset = twilight(timeRange = timeRangeData,
                           latLon    = c(47.494427, 8.716432),
                           timeZone  = targetTimeZone)

# Get manual blind times
# ========================================================================
  data(manualBlindTimes)
  cManualBlindTimes = manualBlindTimes

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

saveMTR(mtr      = mtrData,
        filepath = getwd())

## End(Not run)
```

---

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

Examples

```
## Not run:
#' # Set server, database, and other input 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"
  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)

# Get manual blindtimes
# ==============================================================================
  data("manualBlindTimes")
  cManualBlindTimes = manualBlindTimes

# Make Plot
# ==============================================================================
  timeRangePlot = list(c("2021-01-15 00:00", "2021-01-22 00:00"),
                       c("2021-01-23 00:00", "2021-01-31 00:00"))
  cPlot = plotExploration(echoData         = dbData$echoData,
                          timeRange        = timeRangePlot,
                          targetTimeZone   = "Etc/GMT0",
                          manualBlindTimes = cManualBlindTimes,
                          visibilityData   = dbData$visibilityData,
                          protocolData     = dbData$protocolData,
                          sunriseSunset    = dbData$sunriseSunset,
                          maxAltitude      = -1,
                          filePath         = "./")

# Save plot
# ==============================================================================
```

```
  savePlotToFile(plot              = cPlot,
                 filePath          = "./",
                 plotType          = "S",
                 plotWidth_mm      = 400,
                 plotHeight_mm     = 200)

## End(Not run)
```

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

Examples

```
## Not run:
sunrisesunset = twilight(timeRange = c("2021-01-15 00:00",
                                       "2021-01-31 00:00"),
                         latLon    = c(47.494427, 8.716432),
                         timeZone  = "Etc/GMT0")

## End(Not run)
```

Index