Package: motus (via r-universe)

September 2, 2024

Type Package

Version 6.1.1

Title Fetch and use data from the Motus Wildlife Tracking System

Description Retrieve, filter, and visualize telemetry data from the Motus Wildlife Tracking System <https://motus.org>.

URL https://motuswts.github.io/motus/

BugReports https://github.com/MotusWTS/motus/issues

Depends R (>= 3.2.0)

Imports DBI (>= 1.0.0), dbplyr (>= 2.1.1), dplyr (>= 1.0.4), httr (>= 1.4.0), geosphere (>= 1.5.7), ggplot2 (>= 3.2.0), ggspatial (>= 1.1.9), glue (>= 1.4.2), gridExtra (>= 2.3), jsonlite (>= 1.6), lubridate (>= 1.7.4), lutz (>= 0.3.1), magrittr, methods, purrr (>= 0.3.0), rlang (>= 0.3.1), RSQLite (>= 2.1.1), stringr (>= 1.4.0), tidyr (>= 0.8.3), tidyselect (>= 1.0.0), suncalc

Suggests covr, mockery, prettymapr, rmarkdown, roxygen2, sf (>= 1.0.14), spelling, testthat (>= 3.0.0), withr

License GPL-3

RoxygenNote 7.2.3

Roxygen list(markdown = TRUE)

Encoding UTF-8

Language en-CA

LazyData true

Config/testthat/edition 3

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

RemoteUrl https://github.com/MotusWTS/motus

RemoteRef HEAD

RemoteSha 73fecbf7ebdc2e18a1850fc172326272a73a3b34

Contents

activity — *Add/update batch activity*

Description

Download or resume a download of the `activity` table in an existing Motus database. Batch activity refers to the number of hits detected during a given batch. Batches with large numbers of hits may indicate interference and thus unreliable hits.

Usage

```
activity(src, resume = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`resume`	Logical. Resume a download? Otherwise the table is removed and the download is started from the beginning.

Details

This function is automatically run by the `tagme()` function with `resume = TRUE`.

If an `activity` table doesn't exist, it will be created prior to downloading. If there is an existing `activity` table, this will update the records.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Access 'activity' table
library(dplyr)
a <- tbl(sql_motus, "activity")

# If interrupted and you want to resume
## Not run: my_tags <- activity(sql_motus, resume = TRUE)
```

activityAll *Add/update all batch activity*

Description

Download or resume a download of the `activityAll` table in an existing Motus database. Batch activity refers to the number of hits detected during a given batch. Batches with large numbers of hits may indicate interference and thus unreliable hits.

Usage

```
activityAll(src, resume = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`resume`	Logical. Resume a download? Otherwise the table is removed and the download is started from the beginning.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Get all activity
## Not run: sql_motus <- activityAll(sql_motus)

# Access 'activityAll' table
library(dplyr)
a <- tbl(sql_motus, "activityAll")

# If interrupted and you want to resume
## Not run: sql_motus <- activityAll(sql_motus, resume = TRUE)
```

checkVersion *Check database version*

Description

Verifies the version of the package against the `admInfo` table of a `.motus` file. Those should match if the `updateMotusDb()` function has been properly applied by the `tagme()` function.

Usage

```
checkVersion(src)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.

`clarify` *Report or claim ambiguous tag detections*

Description

A detections is "ambiguous" if the motus tag finder could not tell which of several tags was detected, because they all produce the same signal and were active at the same time. The motus tag finder uses tag deployment and lifetime metadata to decide what tags to seek when, and notices when it can't distinguish between two or more of them. Detections of such tags during these periods of overlap are assigned a negative motus tag ID that represents from 2 to 6 possible real motus tags. The ambiguities might be real (i.e. two or more tags transmitting the same signal and active at the same time), or due to errors in tag registration or deployment metadata.

Usage

```
clarify(src, id, from, to, all.mine = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`id`	if not missing, a vector of negative motus ambiguous tag IDs for which you wish to claim detections. If missing, all tags are claimed over any period specified by `from` and `to`.
`from`	Character. If not missing, the start time for your claim to ambiguous detections of tag(s) `id`. If missing, you are claiming all detections up to `to`. `from` can be a numeric timestamp, or a character string compatible with `lubridate::ymd()`
`to`	Character. If not missing, the end time for your claim to ambiguous detections of tag(s) `id`. If missing, you are claiming all detections after `from`. `to` can be a numeric timestamp, or a character string compatible with `lubridate::ymd()`
`all.mine`	Logical. If TRUE, claim all ambiguous detections. In this case, `id`, `from` and `to` are ignored.

Details

This function serves two purposes:

- called with only a database, it reports the numbers of ambiguous detections and what they could represent.
- called with `id`, it lets you claim some of the ambiguities as your own tag, so that in subsequent processing, they will appear to be yours.

This function does not (yet?) report your claim to motus.org

WARNING: you cannot undo a claim within a copy of the database. If unsure, copy the .motus file first, then run `clarify` on only one copy.

If both `from` and `to` are missing, then all detections of ambiguous tag(s) `id` are claimed.

Parameters `id`, `from`, and `to` are recycled to the length of the longest item.

When you claim an ambiguous tag `T` for a period, any runs of `T` which overlap that period at all are claimed entirely, even if they extend beyond the period; i.e. runs are not split.

Value

With no parameters, returns a summary data frame of ambiguous tag detections

Examples

```
## Not run:
s <- tagme(57)         # get the tag database for project 57
clarify(s)             # report on the ambiguous tag detections in s
clarify(all.mine = TRUE) # claim all ambiguous tag detections as mine
clarify(id = -57)      # claim all detections of ambiguous tag -57 as mine

clarify(id = c(-72, -88, -91), from = "2017-01-02", to = "2017-05-06")
# claim all detections of ambiguous tags -72, -88, and -91 from
#   January 2 through May 6, 2017, as mine

## End(Not run)
```

`createRunsFilter` *Create a new filter records that can be applied to runs*

Description

Create a new filter records that can be applied to runs

Usage

```
createRunsFilter(src, filterName, motusProjID = NA, descr = NA, update = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`filterName`	Character. Unique name given to the filter
`motusProjID`	Character. Optional project ID attached to the filter in order to share with other users of the same project.
`descr`	Character. Optional filter description detailing what the filter is meant to do
`update`	Logical. Whether the filter record gets updated when a filter with the same name already exists.

Value

an integer `filterID`

`deleteRunsFilter` *Delete a filter*

Description

Deletes a filter by name or project ID.

Usage

```
deleteRunsFilter(src, filterName, motusProjID = NA, clearOnly = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`filterName`	Character. Unique name given to the filter
`motusProjID`	Character. Optional project ID attached to the filter in order to share with other users of the same project.
`clearOnly`	Logical. When true, only remove the probability records associated with the filter, but retain the filter itself

Value

the integer `filterID` of the filter deleted

deprecateBatches *Fetch and remove deprecated batches*

Description

Deprecated batches are removed from the online database but not from local data files. This function fetches a list of deprecated batches (stored in the 'deprecated' table), and, optionally, removes these batches from all tables that reference `batchID`s

Usage

```
deprecateBatches(src, fetchOnly = FALSE, ask = TRUE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`fetchOnly`	Logical. Only *fetch* batches that are deprecated. Don't remove deprecated batches from other tables.
`ask`	Logical. Ask for confirmation when removing deprecated batches

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run:
sql_motus <- tagme(176, new = TRUE)

# Access 'deprecated' table using tbl() from dplyr
library(dplyr)
tbl(sql_motus, "deprecated")

# See that there are deprecated batches in the data
filter(tbl(sql_motus, "alltags"), batchID == 6000)

# Fetch deprecated batches
deprecateBatches(sql_motus, fetchOnly = TRUE)

# Remove deprecated batches (will ask for confirmation unless ask = FALSE)
deprecateBatches(sql_motus, ask = FALSE)

# See that there are NO more deprecated batches in the data
filter(tbl(sql_motus, "alltags"), batchID == 6000)

## End(Not run)
```

`filterByActivity` *Filter* `alltags` *by* `activity`

Description

The `activity` table is used to identify batches with too much noise. Depending on the value of `return` these are filtered out, returned, or identified in the `alltags` view with the column `probability`. **No changes to the database are made.**

Usage

```
filterByActivity(
  src,
  return = "good",
  view = "alltags",
  minLen = 3,
  maxLen = 5,
  maxRuns = 100,
  ratio = 0.85
)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`return`	Character. One of "good" (return only 'good' runs), "bad" (return only 'bad' runs), "all" (return all runs, but with a new `probability` column which identifies 'bad' (0) and 'good' (1) runs.
`view`	Character. Which view to use, one of "alltags" (faster) or "alltagsGPS" (with GPS data).
`minLen`	Numeric. The minimum run length to allow (equal to or below this, all runs are 'bad')
`maxLen`	Numeric. The maximum run length to allow (equal to or above this, all runs are 'good')
`maxRuns`	Numeric. The cutoff of number of runs in a batch (see Details)
`ratio`	Numeric. The ratio cutoff of runs length 2 to number of runs in a batch (see Details)

Details

Runs are identified by the following:

- All runs with a length >= `maxLen` are **GOOD**
- All runs with a length <= `minLen` are **BAD**
- Runs with a length between `minLen` and `maxLen` are **BAD** IF both of the following is true:
  - belong to a batch where the number of runs is >= `maxRuns`
  - the ratio of runs with a length of 2 to the number of runs total is >= `ratio`

Value

tbl_SQLiteConnection

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

tbl_good <- filterByActivity(sql_motus)
tbl_bad <- filterByActivity(sql_motus, return = "bad")
tbl_all <- filterByActivity(sql_motus, return = "all")
```

`getAccess` *Return accessible projects and receivers*

Description

Return the projects and receivers which are accessible by the given credentials

Usage

```
getAccess()
```

Examples

```
## Not run:
getAccess()

## End(Not run)
```

`getGPS` *Get GPS variables*

Description

To improve speed, the `alltags` view doesn't include GPS-related variables such as `gpsLat`, `gpsLon`, or `gpsAlt`. There is a `alltagsGPS` view that does include GPS-related variables, but this will take time to load. This function accepts a source and returns the GPS data associated with the `hitIDs` in the `alltags` view. Optionally, users can supply a subset of the `alltags` view to return only GPS data associated with the specific `hitIDs` present in the subset.

Usage

```
getGPS(src, data = NULL, by = "daily", cutoff = NULL, keepAll = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`data`	SQLite connection or data.frame. Optional subset of the `alltags` view. Must have `ts`, `batchID` and `hitID` at the minimum.
`by`	Numeric/Character. Either the time in minutes over which to join GPS locations to hits, or "daily" or "closest". To join GPS locations by daily time blocks or by the closest temporal match (see Details).
`cutoff`	Numeric. The maximum allowable time in minutes between hit and GPS timestamps when matching hits to GPS with `by = 'closest'`. Defaults to `NULL` (no maximum).
`keepAll`	Logical. Return all hits regardless of whether they have a GPS match? Defaults to FALSE.

Details

There are three different methods for matching GPS data to `hitID`s all related to timestamps (`ts`).

1. `by = X` Where `X` is a duration in minutes. `ts` is converted to a specific time block of duration `X`. Median GPS lat/longs for the time block are returned, matching associated `hitID` time blocks.
2. `by = "daily"` (the default). Similar to `by = X` except the duration is 24hr.
3. `by = "closest"` Individual GPS lat/lons are returned, matching the closest `hitID` timestamp. Use `cutoff` to specify the maximum allowable time between timestamps (defaults to none).

Value

Data frame linking hitID to gpsLat, gpsLon and gpsAlt. When `by = 'daily'` or `by = 'X'`, output includes:

- `hitID` - the ID associated with the hit
- `gpsLat` \ `gpsLon` \ `gpsAlt` - the median location calculated from the available GPS points
- `gpsTs_min \ gps_Ts_max` - the range of GPS timestamps associated with the GPS points binned

When `by = 'closest'` or `by = 'X'`, output includes:

- `hitID` - the ID associated with the hit
- `gpsID` - the ID of the closest GPS point aligned with the `hitID`
- `gpsLat` \ `gpsLon` \ `gpsAlt` - the location of the GPS point
- `gpsTs` - the timestamp of the GPS point

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Match hits to GPS within 24hrs (daily) of each other
my_gps <- getGPS(sql_motus)
my_gps

# Note that the sample data doesn't have GPS hits so this will be an
# empty data frame for project 176.

# Match hits to GPS within 15min of each other
my_gps <- getGPS(sql_motus, by = 15)
my_gps

# Match hits to GPS according to the closest timestamp
my_gps <- getGPS(sql_motus, by = "closest")
my_gps

# Match hits to GPS according to the closest timestamp, but limit to within
# 20min of each other
my_gps <- getGPS(sql_motus, by = "closest", cutoff = 20)
my_gps

# To return all hits, regardless of whether they match a GPS record

my_gps <- getGPS(sql_motus, keepAll = TRUE)
my_gps

# Alternatively, use the alltagsGPS view:
dplyr::tbl(sql_motus, "alltagsGPS")
```

`getMotusDBSrc` *Get the src_sqlite for a receiver or tag database*

Description

Receiver database files have names like "SG-1234BBBK06EA.motus" or "Lotek-12345.motus", and project database files have names like "project-52.motus".

Usage

```
getMotusDBSrc(
  recv = NULL,
  proj = NULL,
  create = FALSE,
  dbDir = motus_vars$dbDir
)
```

Arguments

`recv`	receiver serial number
`proj`	integer motus project number exactly one of `proj` or `recv` must be specified.
`create`	Is this a new database? Default: FALSE. Same semantics as for `src_sqlite()`'s parameter of the same name: the DB must already exist unless you specify `create = TRUE`
`dbDir`	path to folder with existing receiver databases Default: `motus_vars$dbDir`, which is set to the current folder by `getwd()` when this library is loaded.

Value

a src_sqlite for the receiver; if the receiver is new, this database will be empty, but have the correct schema.

`getRuns` *Returns a dataframe containing runs*

Description

Specifically the `runID` and `motusTagID`, `ambigID` and `tsBegin` to `tsEnd` (timestamp) range of runs, filtered by optional parameters. The `match.partial` parameter (default = TRUE) determines how timestamp filtering works. When `match.partial` is FALSE, `runID`'s are only included when both `tsBegin` and `tsEnd` falls between `ts.min` and `ts.max` (only includes runs when they entirely contained in the specified range). When match.partial is TRUE, `runID`'s are returned whenever the run partially matches the specified period.

Usage

```
getRuns(
  src,
  ts.min = NA,
  ts.max = NA,
  match.partial = TRUE,
  motusTagID = c(),
  ambigID = c()
)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`ts.min`	minimum timestamp used to filter the dataframe, Default: NA

`ts.max`	maximum timestamp used to filter the dataframe, Default: NA
`match.partial`	whether runs that partially overlap the specified ts range are included, Default: TRUE
`motusTagID`	vector of Motus tag ID's used to filter the resulting dataframe, Default: c()
`ambigID`	vector of ambig ID's used to filter the resulting dataframe, Default: c()

Value

a dataframe containing the runID, the motusTagID and the ambigID (if applicable) of runs

`getRunsFilters` *Get runsFilters*

Description

Returns a dataframe of the `runsFilters` records matching a filter name (and optionally a project ID) stored in the local database.

Usage

```
getRunsFilters(src, filterName, motusProjID = NA)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`filterName`	Character. Unique name given to the filter
`motusProjID`	Character. Optional project ID attached to the filter in order to share with other users of the same project.

Value

a database connection to `src`

`gpsAll` *Add/update all GPS points*

Description

Download or resume a download of the `gpsAll` table in an existing Motus database. Batch activity refers to the number of hits detected during a given batch. Batches with large numbers of hits may indicate interference and thus unreliable hits.

Usage

```
gpsAll(src, resume = TRUE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`resume`	Logical. Resume a download? Otherwise the table is removed and the download is started from the beginning.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Get all GPS points
## Not run: sql_motus <- gpsAll(sql_motus)

# Access 'gpsAll' table
library(dplyr)
g <- tbl(sql_motus, "gpsAll")

# gpsAll resumes a previous download by default
# If you want to delete this original data and do a fresh download,
# use resume = FALSE
## Not run: sql_motus <- gpsAll(sql_motus, resume = FALSE)
```

`listRunsFilters` *Returns a dataframe of the filters stored in the local database.*

Description

Returns a dataframe of the filters stored in the local database.

Usage

```
listRunsFilters(src)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.

Value

a dataframe

`metadata` *Update all metadata*

Description

Updates the entire metadata for receivers and tags from Motus server. Contrary to `tagme()`, this function retrieves the entire set of metadata for tags and receivers, and not only those pertinent to the detections in your local file.

Usage

```
metadata(src, projectIDs = NULL, replace = TRUE, delete = FALSE)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`projectIDs`	optional integer vector of Motus projects IDs for which metadata should be obtained; default: NULL, meaning obtain metadata for all tags and receivers that your permissions allow.
`replace`	logical scalar; if TRUE (default), existing data replace the existing metadata with the newly acquired ones.
`delete`	logical scalar; Default = FALSE. if TRUE, the entire metadata tables are cleared (for all projects) before re-importing the metadata.

See Also

`tagme()` provides an option to update only the metadata relevant to a specific project or receiver file.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Add extended metadata to your file
## Not run: metadata(sql_motus)

# Access different metadata tables
library(dplyr)
tbl(sql_motus, "species")
tbl(sql_motus, "projs")
tbl(sql_motus, "tagDeps")
# Etc.
```

`motus` *Fetch and use data from the Motus Wildlife Tracking System*

Description

`motus` is an R package for retrieving telemetry data from the Motus Wildlife Tracking System `https://motus.org`.

Details

For a detailed walk-though and instructions check out the walk-throughs and articles!

Commonly used functions:

1. Download telemetry data
   - `tagme()`
   - `tellme()`
   - `metadata()`
   - `checkVersion()`
2. Create data filters
   - `listRunsFilters()`
   - `getRunsFilters()`
   - `createRunsFilter()`
   - `writeRunsFilter()`

   - deleteRunsFilter()
3. Summarize data
   - tagSum()
   - tagSumSite()
   - simSiteDet()
   - siteSum()
   - siteSumDaily()
   - siteTrans()
4. Plot data
   - plotAllTagsCoord()
   - plotAllTagsSite()
   - plotDailySiteSum()
   - plotRouteMap()
   - plotSite()
   - plotSiteSig()
   - plotTagSig()
5. Sunrises and sets
   - sunRiseSet()
   - timeToSunriset()

References

Motus Wildlife Tracking System https://motus.org

---

motusLogout *Forget login credentials for motus.*

Description

Any requests to the motus data server after calling this function will require re-entering a username and password.

Usage

```
motusLogout()
```

Details

This function just resets these items to NULL:

- motus_vars$authToken
- motus_vars$userLogin
- motus_vars$userPassword

Due to their active bindings, subsequent calls to any functions that need them will prompt for a login.

Value

TRUE.

---

nodeData *Add/update nodeData*

Description

Download or resume a download of the 'nodeData' table in an existing Motus database. nodeData contains information regarding the 'health' of portable node units.

Usage

```
nodeData(src, resume = FALSE)
```

Arguments

src	SQLite connection. Result of tagme(XXX) or DBI::dbConnect(RSQLite::SQLite(), "XXX.motus").
resume	Logical. Resume a download? Otherwise the table is removed and the download is started from the beginning.

Details

This function is automatically run by the tagme() function with resume = TRUE.

If an nodeData table doesn't exist, it will be created prior to downloading. If there is an existing nodeData table, this will update the records.

Note that only records for CTT tags will have the possibility of nodeData.

Node metadata is found in the nodeDeps table, updated along with other metadata.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Access `nodeData` table
library(dplyr)
a <- tbl(sql_motus, "nodeData")

# If you just want to download `nodeData`
## Not run: my_tags <- nodeData(sql_motus)
```

plotAllTagsCoord *Plot all tag detections by latitude or longitude*

Description

Plot latitude/longitude vs time (UTC rounded to the hour) for each tag using motus detection data. Coordinate is by default taken from a receivers deployment latitude in metadata.

Usage

```
plotAllTagsCoord(
  data,
  coordinate = "recvDeployLat",
  ts = "ts",
  tagsPerPanel = 5
)
```

Arguments

`data`	a selected table from motus data, eg. "alltags", or a data.frame of detection data including at a minimum variables for `recvDeployName`, `fullID`, `mfgID`, date/time, `latitude` or `longitude`
`coordinate`	column name from which to obtain location values, by default it is set to `recvDeployLat`
`ts`	Character. Name of column with timestamp values, defaults to `ts`.
`tagsPerPanel`	number of tags in each panel of the plot, by default this is 5

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltags")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Plot tbl file tbl_alltags with default GPS latitude data and 5 tags per panel
plotAllTagsCoord(tbl_alltags)

# Plot an sql file tbl_alltags with 10 tags per panel
plotAllTagsCoord(tbl_alltags, tagsPerPanel = 10)
```

```
# Plot dataframe df_alltags using receiver deployment latitudes with default
# 5 tags per panel
plotAllTagsCoord(df_alltags, coordinate = "recvDeployLat")

# Plot dataframe df_alltags using LONGITUDES and 10 tags per panel
# But only works if non-NA "gpsLon"!
## Not run: plotAllTagsCoord(df_alltags, coordinate = "gpsLon", tagsPerPanel = 10)

# Plot dataframe df_alltags using lat for select motus tagIDs
plotAllTagsCoord(filter(df_alltags, motusTagID %in% c(19129, 16011, 17357)),
                 tagsPerPanel = 1)
```

`plotAllTagsSite` *Plot all tag detections by deployment*

Description

Plot deployment (ordered by latitude) vs time (UTC) for each tag

Usage

```
plotAllTagsSite(data, coordinate = "recvDeployLat", tagsPerPanel = 5)
```

Arguments

`data`	a selected table from .motus data, eg. "alltags", or a data.frame of detection data including at a minimum variables for `recvDeployName`, `fullID`, `mfgID`, date/time, `latitude` or `longitude`
`coordinate`	column of receiver latitude/longitude values to use, defaults to `recvDeployLat`
`tagsPerPanel`	number of tags in each panel of the plot, default is 5

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltags")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()
```

```
# Plot detections of dataframe df_alltags by site ordered by latitude, with
# default 5 tags per panel
plotAllTagsSite(df_alltags)

# Plot detections of dataframe df_alltags by site ordered by latitude, with
# 10 tags per panel
plotAllTagsSite(df_alltags, tagsPerPanel = 10)

# Plot detections of tbl file tbl_alltags by site ordered by receiver
# deployment latitude
plotAllTagsSite(tbl_alltags, coordinate = "recvDeployLon")

# Plot tbl file tbl_alltags using 3 tags per panel for species Red Knot
plotAllTagsSite(filter(tbl_alltags, speciesEN == "Red Knot"), tagsPerPanel = 3)
```

plotDailySiteSum *Plots number of detections and tags, daily, for a specified site*

Description

Plots total number of detections across all tags, and total number of tags detected per day for a specified site. Depends on `siteSumDaily()`.

Usage

```
plotDailySiteSum(data, recvDeployName)
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for `motusTagID`, `sig`, `recvDeployName`, `ts`
`recvDeployName`	name of site to plot

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Plot of all tag detections at site Longridge using dataframe df_alltags
plotDailySiteSum(df_alltags, recvDeployName = "Longridge")

# Plot of all tag detections at site Niapiskau using tbl file tbl_alltags
plotDailySiteSum(df_alltags, recvDeployName = "Niapiskau")
```

---

plotRouteMap *Map of tag routes and sites coloured by id*

Description

Google map of routes of Motus tag detections coloured by ID. User defines a date range to show points for receivers that were operational at some point during the date range.

Usage

```
plotRouteMap(
  src,
  maptype = "osm",
  zoom = NULL,
  start_date = NULL,
  end_date = NULL,
  lim_lat = NULL,
  lim_lon = NULL,
  data,
  lat,
  lon,
  recvStart,
  recvEnd
)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`maptype`	Character. Map tiles to use. Must be one of `rosm::osm.types()`, such as `osm`, `stamenbw`, etc. Most map tiles require attribution for publication, see details.
`zoom`	Integer. Override the calculated zoom level to increase or decrease the resolution of the map tiles.
`start_date`	Character. Optional start date for routes.
`end_date`	Character. Optional end date for routes.
`lim_lat`	Numeric vector. Optional latitudinal plot limits.
`lim_lon`	Numeric vector. Optional longitudinal plot limits.
`data`	Defunct, use `src`, `df_src`, or `df` instead.

`lat`	Defunct
`lon`	Defunct
`recvStart`	Defunct
`recvEnd`	Defunct

Details

By default this function uses OSM maps (Open Street Map). OSM and many other map tiles are released under specific licences, which generally require that you give attribution at a minimum. See OSM for more details on their tiles, but remember to check what other groups require if you use their tiles.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Plot route map of all detection data, with "osm" maptype, and receivers
# active between 2016-01-01 and 2017-01-01
plotRouteMap(sql_motus, start_date = "2016-01-01", end_date = "2016-12-31")
```

`plotSite` *Plot all tags by site*

Description

Plot tag ID vs time for all tags detected by site, coloured by antenna bearing. Input is expected to be a data frame, database table, or database. The data must contain "ts", "antBearing", "fullID", "recvDeployName", "recvDeployLat", "recvDeployLon", and optionally "gpsLat" and "gpsLon". If GPS lat/lon are included, they will be used rather than recvDeployLat/Lon. These data are generally contained in the `alltags` or the `alltagsGPS` views. If a motus database is submitted, the `alltagsGPS` view will be used.

Usage

```
plotSite(df_src, sitename = NULL, ncol = NULL, nrow = NULL, data)
```

Arguments

`df_src`	Data frame, SQLite connection, or SQLite table. An SQLite connection would be the result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`; an SQLite table would be the result of `dplyr::tbl(tags, "alltags")`; a data frame could be the result of `dplyr::tbl(tags, "alltags") %>% dplyr::collect()`.

`sitename`	Character vector. Subset of sites to plot. If `NULL`, all unique sites are plotted.
`ncol`	Numeric. Passed on to `ggplot2::facet_wrap()`
`nrow`	Numeric. Passed on to `ggplot2::facet_wrap()`
`data`	Defunct, use `src`, `df_src`, or `df` instead.

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# Plot all sites within file for tbl file tbl_alltags
plotSite(tbl_alltags)

# Plot only detections at a specific site; Piskwamish
plotSite(tbl_alltags, sitename = "Piskwamish")

# For more custom filtering, convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- collect(tbl_alltags)

# Plot only detections for specified tags for data.frame df_alltags
plotSite(filter(df_alltags, motusTagID %in% c(16047, 16037, 16039)))
```

`plotSiteSig` *Plot signal strength of all tags by a specified site*

Description

Plot signal strength vs time for all tags detected at a specified site, coloured by antenna

Usage

```
plotSiteSig(data, recvDeployName)
```

Arguments

`data`	a selected table from .motus data, eg. "alltags", or a data.frame of detection data including at a minimum variables for `antBearing`, `ts`, `recvDeployLat`, `sig`, `fullID`, `recvDeployName`
`recvDeployName`	name of `recvDeployName`

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltags")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Plot all tags for site Piskwamish
plotSiteSig(tbl_alltags, recvDeployName = "Piskwamish")

# Plot select tags for site Piskwamish
plotSiteSig(filter(df_alltags, motusTagID %in% c(16037, 16039, 16035)),
  recvDeployName = "Netitishi")
```

---

plotTagSig *Plot signal strength of all detections for a specified tag by site*

Description

Plot signal strength vs time for specified tag, faceted by site (ordered by latitude) and coloured by antenna

Usage

```
plotTagSig(data, motusTagID)
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for motusTagID, sig, ts, antBearing, recvDeployLat, fullID, recvDeployName, gpsLat, gpsLon
`motusTagID`	a numeric motusTagID to display in plot

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()
```

```
# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Plot signal strength of a specified tag using dataframe df_alltags
plotTagSig(df_alltags, motusTagID = 16047)

# Plot signal strength of a specified tag using tbl file tbl_alltags
plotTagSig(tbl_alltags, motusTagID = 16035)
```

points2Path *Convert points to path*

Description

Converts a data frame with a list of lat/lons to a spatial data frame with MULTILINES defining paths by tag id. Useful for plotting with `ggplot2::geom_sf()`. Will silently remove single points.

Usage

```
points2Path(df, by = "fullID", lat = "recvDeployLat", lon = "recvDeployLon")
```

Arguments

`df`	Data frame. Could be the result of `dplyr::tbl(tags, "alltags") %>% dplyr::collect()`.
`by`	Character. Column defining the tag id over which to group points into paths. Defaults to "fullID".
`lat`	Character. Name of column with latitude values, defaults to `recvDeployLat`.
`lon`	Character. Name of column with longitude values, defaults to `recvDeployLon`.

Value

Spatial data frame with MULTILINE paths

simSiteDet *Create a dataframe of simultaneous detections at multiple sites*

Description

Creates a dataframe consisting of detections of tags that are detected at two or more receiver at the same time.

Usage

```
simSiteDet(data)
```

Arguments

`data`	a selected table from .motus data, eg. "alltags", or a data.frame of detection data including at a minimum variables for `motusTagID`, `recvDeployName`, `ts`

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltags")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# To get a data.frame of just simultaneous detections from a tbl file
# tbl_alltags
simSites <- simSiteDet(tbl_alltags)

# To get a data.frame of just simultaneous detections from a dataframe
# df_alltags
simSites <- simSiteDet(df_alltags)
```

`siteSum` *Summarize and plot detections of all tags by site*

Description

Creates a summary of the first and last detection at a site, the length of time between first and last detection, the number of tags, and the total number of detections at a site. Plots total number of detections across all tags, and total number of tags detected at each site.

Usage

```
siteSum(data, units = "hours")
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for motusTagID, sig, recvDeployLat, recvDeployLon, recvDeployName, ts, gpsLat, and gpsLon
`units`	units to display time difference, defaults to "hours", options include "secs", "mins", "hours", "days", "weeks"

Value

a data.frame with these columns:

- site: site
- first_ts: time of first detection at specified site
- last_ts: time of last detection at specified site
- tot_ts: total amount of time between first and last detection at specified site, output in specified unit (defaults to "hours")
- num.tags: total number of unique tags detected at specified site
- num.det: total number of tag detections at specified site

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Create site summaries for all sites within detection data with time in
# default hours using data.frame df_alltags
site_summary <- siteSum(tbl_alltags)

# Create site summaries for only select sites with time in minutes
sub <- filter(df_alltags, recvDeployName %in%
                c("Niapiskau", "Netitishi", "Old Cur", "Washkaugou"))
site_summary <- siteSum(sub, units = "mins")

# Create site summaries for only a select species, Red Knot
site_summary <- siteSum(filter(df_alltags, speciesEN == "Red Knot"))
```

`siteSumDaily` *Summarize daily detections of all tags by site*

Description

Creates a summary of the first and last daily detection at a site, the length of time between first and last detection, the number of tags, and the total number of detections at a site for each day. Same as `siteSum()`, but daily by site.

Usage

```
siteSumDaily(data, units = "hours")
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for `motusTagID`, `sig`, `recvDeployName`, `ts`
`units`	units to display time difference, defaults to "hours", options include "secs", "mins", "hours", "days", "weeks"

Value

a data.frame with these columns:

- recvDeployName: site name of deployment
- date: date that is being summarized
- first_ts: time of first detection on specified "date" at "recvDeployName"
- last_ts: time of last detection on specified "date" at "recvDeployName"
- tot_ts: total amount of time between first and last detection at "recvDeployName" on "date, output in specified unit (defaults to "hours")
- num.tags: total number of unique tags detected at "recvDeployName", on "date"
- num.det: total number of detections at "recvDeployName", on "date"

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags %>%
  collect() %>%
  as.data.frame()

# Create site summaries for all sites within detection data with time in
# minutes using tbl file tbl_alltags
daily_site_summary <- siteSumDaily(tbl_alltags, units = "mins")

# Create site summaries for only select sites with time in minutes using tbl
# file tbl_alltags
sub <- filter(tbl_alltags, recvDeployName %in% c("Niapiskau", "Netitishi",
                                                  "Old Cut", "Washkaugou"))
daily_site_summary <- siteSumDaily(sub, units = "mins")

# Create site summaries for only a select species, Red Knot, with default
# time in hours using data frame df_alltags
daily_site_summary <- siteSumDaily(filter(df_alltags,
                                          speciesEN == "Red Knot"))
```

`siteTrans` *Summarize transitions between sites for each tag*

Description

Creates a dataframe of transitions between sites; detections are ordered by detection time, then "transitions" are identified as the period between the final detection at site x (possible "departure"), and the first detection (possible "arrival") at site y (ordered chronologically). Each row contains the last detection time and lat/lon of site x, first detection time and lat/lon of site y, distance between the site pair, time between detections, rate of movement between detections, and bearing between site pairs.

Usage

```
siteTrans(data, latCoord = "recvDeployLat", lonCoord = "recvDeployLon")
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for `ts`, `motusTagID`, `tagDeployID`, `recvDeployName`, and a latitude/longitude
`latCoord`	a variable with numeric latitude values, defaults to `recvDeployLat`
`lonCoord`	a variable with numeric longitude values, defaults to `recvDeployLon`

Value

a data.frame with these columns:

- fullID: fullID of Motus registered tag
- ts.x: time of last detection of tag at site.x ("departure" time)
- lat.x: latitude of site.x
- lon.x: longitude of site.x
- site.x: first site in transition pair (the "departure" site)
- ts.y: time of first detection of tag at site.y ("arrival" time)
- lat.y: latitude of site.y
- lon.y: longitude of site.y
- site.y: second site in transition pair (the "departure" site)
- tot_ts: length of time between ts.x and ts.y (in seconds)
- dist: total straight line distance between site.x and site.y (in metres), see `sensorgnome::latLonDist()` for details
- rate: overall rate of movement (tot_ts/dist), in metres/second
- bearing: bearing between first and last detection sites, see bearing function in geosphere package for more details

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

## convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
 df_alltags <- tbl_alltags %>%
   collect() %>%
   as.data.frame()

# View all site transitions for all detection data from tbl file tbl_alltags
transitions <- siteTrans(tbl_alltags)
```

```
# View site transitions for only tag 16037 from data.frame df_alltags using
# gpsLat/gpsLon
transitions <- siteTrans(filter(df_alltags, motusTagID == 16037),
                         latCoord = "gpsLat", lonCoord = "gpsLon")
```

srvTimeout *Sets global options for timeouts*

Description

Sets, resets or returns the "motus.timeout" global option used by all API access functions (including `tagme()`). If `timeout` is a number and `reset` is `FALSE`, the API timeout is set to `timeout` number of seconds. If `reset` is `TRUE`, the API timeout is reset to the default of 120 seconds. If no `timeout` is defined and `reset = FALSE`, the current value of the timeout is returned.

Usage

```
srvTimeout(timeout, reset = FALSE)
```

Arguments

`timeout`	Numeric. Number of seconds to wait for a response from the server. Increase if you're working with a project that requires extra time to process and serve the data.
`reset`	Logical. Whether to reset the timeout to the default (120s; default `FALSE`). If `TRUE`, `timeout` is ignored.

Details

By default the timeout is 120s, which generally should give the server sufficient time to prepare the data without having the user wait for too long if the API is unavailable. However, some projects take unusually long to compile the data, so a longer timeout may be warranted in those situations. This is equivalent to `options(motus.timeout = timeout)`

Value

Nothing. Or, if `timeout` is missing and `reset = FALSE`, the current timeout value.

Examples

```
srvTimeout()   # get the timeout value
srvTimeout(5)  # set the timeout value
srvTimeout()   # get the timeout value

## Not run:
# No problem with default timeouts
t <- tagme(176, new = TRUE)
```

```
# But setting the timeout too short results in a server timeout
srvTimeout(0.001)
t <- tagme(176, new = TRUE)

## End(Not run)
```

---

sunRiseSet *Obtain sunrise and sunset times*

Description

Create and add sunrise and sunset columns to tag data. Can take a motus database table, but will always return a collected data frame. Requires data containing at least latitude, longitude, and time.

Usage

```
sunRiseSet(
  df_src,
  lat = "recvDeployLat",
  lon = "recvDeployLon",
  ts = "ts",
  data
)
```

Arguments

`df_src`	Data frame, SQLite connection, or SQLite table. An SQLite connection would be the result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`; an SQLite table would be the result of `dplyr::tbl(tags, "alltags")`; a data frame could be the result of `dplyr::tbl(tags, "alltags") %>% dplyr::collect()`.
`lat`	Character. Name of column with latitude values, defaults to `recvDeployLat`.
`lon`	Character. Name of column with longitude values, defaults to `recvDeployLon`.
`ts`	Character. Name of column with timestamp values, defaults to `ts`.
`data`	Defunct, use `src`, `df_src`, or `df` instead.

Details

Note that this will always return the sunrise and sunset of the *local* date. For example, 2023-01-01 04:00:00 in Central North American time is 2023-01-01 in UTC, but 2023-01-01 20:00:00 is actually the following date in UTC. Because Motus timestamps are UTC, times are first converted to their local time zone time using the lat/lon coordinates before extracting the date. Thus:

- A UTC timestamp of 1672624800 for Winnipeg, Canada is 2023-01-02 02:00:00 UTC and 2023-01-01 20:00:00 local time

- Therefore `sunRiseSet()` calculates the sunrise/sunset times for 2023-01-01 (not for 2023-01-02)
- These sunrise/sunset times are returned in UTC: 2023-01-01 14:27:50 UTC and 2023-01-01 22:38:30 UTC
- Note that the UTC timestamp 2023-01-02 02:00:00 is later than the sunset time of 2023-01-01 22:38:30 UTC. This makes sense, as we know that the timestamp is ~8pm local time, well after sunset in the winter for that date.

Value

Original data (as a flat data frame), with the following additional columns:

- `sunrise` - Time of sunrise in **UTC** for that row's date and location
- `sunset` - Time of sunset in **UTC** for that row's date and location

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# For SQLite Data base---------------------------------------------
sun <- sunRiseSet(sql_motus)

# For specific SQLite table/view ------------------------------------
library(dplyr)
tbl_alltagsGPS <- tbl(sql_motus, "alltagsGPS")
sun <- sunRiseSet(tbl_alltagsGPS)

# For a flattened data frame -----------------------------------------
df_alltagsGPS <- collect(tbl_alltagsGPS)
sun <- sunRiseSet(df_alltagsGPS)

# Using alternate lat/lons ------------------------------------------
# Get sunrise and sunset information from tbl_alltags using gps lat/lon
# Note this will only work if there are non-NA values in gpsLat/gpsLon
## Not run: sun <- sunRiseSet(tbl_alltagsGPS, lat = "gpsLat", lon = "gpsLon")
```

`tagme` *Download motus tag detections to a database*

Description

This is the main motus function for accessing and updating your data. This function downloads motus data to a local SQLite data base in the name of `project-XXX.motus` or `RECIVER_NAME.motus`. If you are having trouble with a particular data base timing out on downloads, see `srvTimeout()` for options.

Usage

```
tagme(
  projRecv,
  update = TRUE,
  new = FALSE,
  dir = getwd(),
  countOnly = FALSE,
  forceMeta = FALSE,
  rename = FALSE,
  skipActivity = FALSE,
  skipNodes = FALSE,
  skipDeprecated = FALSE
)
```

Arguments

`projRecv`	Numeric. Project code from motus.org, *or* character receiver serial number.
`update`	Logical. Download and merge new data (Default `TRUE`)?
`new`	Logical. Create a new database (Default `FALSE`)? Specify `new = TRUE` to create a new local copy of the database to be downloaded. Otherwise, it assumes the database already exists, and will stop with an error if it cannot find it in the current directory. This is mainly to prevent inadvertent downloads of large amounts of data that you already have!
`dir`	Character. Path to the folder where you are storing databases IF `NULL` (default), uses current working directory.
`countOnly`	Logical. If `TRUE`, return only a count of items that would need to be downloaded in order to update the database (Default `FALSE`).
`forceMeta`	Logical. If `TRUE`, re-download metadata for tags and receivers, even if we already have them.
`rename`	Logical. If current SQLite database is of an older data version, automatically rename that database for backup purposes and download the newest version. If `FALSE` (default), user is prompted for action.
`skipActivity`	Logical. Skip checking for and downloading `activity`? See `?activity` for more details
`skipNodes`	Logical. Skip checking for and downloading `nodeData`? See `?nodeData` for more details
`skipDeprecated`	Logical. Skip fetching list of deprecated batches stored in `deprecated`. See `?deprecateBatches()` for more details.

Value

a SQLite Connection for the (possibly updated) database, or a data frame of counts if `countOnly = TRUE`.

See Also

`tellme()`, which is a synonym for `tagme(..., countOnly = TRUE)`

Examples

```
## Not run:

# Create and update a local tag database for motus project 14 in the
# current directory

t <- tagme(14, new = TRUE)

# Update and open the local tag database for motus project 14;
# it must already exist and be in the current directory

t <- tagme(14)

# Update and open the local tag database for a receiver;
# it must already exist and be in the current directory

t <- tagme("SG-1234BBBK4567")

# Open the local tag database for a receiver, without
# updating it

t <- tagme("SG-1234BBBK4567", update = FALSE)

# Open the local tag database for a receiver, but
# tell 'tagme' that it is in a specific directory

t <- tagme("SG-1234BBBK4567", dir = "Projects/gulls")

# Update all existing project and receiver databases in the current working
# directory

tagme()

## End(Not run)
```

`tagmeSample` *Create an in-memory copy of sample tags data*

Description

For running examples and testing out motus functionality, it can be useful to work with sample data set. You can download the most up-to-date copy of this data yourself (to `project-176.motus`) with the username and password both "motus.sample".

Usage

```
tagmeSample(db = "project-176.motus")
```

Arguments

db	Character. Name of sample data base to load. The sample data is "project-176.motus".

Details

`sql_motus <- tagme(176, new = TRUE)`

Or you can use this helper function to grab an in-memory copy bundled in this package.

Value

In memory version of the sample database.

Examples

```
# Explore the sample data
tags <- tagmeSample()
dplyr::tbl(tags, "activity")
dplyr::tbl(tags, "alltags")
```

---

tagSum *General summary of detections for each tag*

Description

Creates a summary for each tag of it's first and last detection time (`ts`), first and last detection site, length of time between first and last detection, straight line distance between first and last detection site, rate of movement, and bearing. Lat/lons are taken from `gpsLat`/`gpsLon`, or if missing, from `recvDeployLat`/`recvDeployLon`. Bearing is calculated using the `geosphere::bearing()` function.

Usage

```
tagSum(df_src, data)
```

Arguments

df_src	Data frame, SQLite connection, or SQLite table. An SQLite connection would be the result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`; an SQLite table would be the result of `dplyr::tbl(tags, "alltags")`; a data frame could be the result of `dplyr::tbl(tags, "alltags") %>% dplyr::collect()`.
data	Defunct, use `src`, `df_src`, or `df` instead.

Value

A flat data frame with the following for each tag:

- `fullID` - `fullID` of Motus registered tag
- `first_ts` - Time (`ts`) of first detection
- `last_ts` - Time (`ts`) of last detection
- `first_site` - First detection site (`recvDeployName`)
- `last_site` - Last detection site (`recvDeployName`)
- `recvLat.x` - Latitude of first detection site (`gpsLat` or `recvDeployLat`)
- `recvLon.x` - Longitude of first detection site (`gpsLon` or `recvDeployLon`)
- `recvLat.y` - Latitude of last detection site (`gpsLat` or `recvDeployLat`)
- `recvLon.y` - Longitude of last detection site (`gpsLon` or `recvDeployLon`)
- `tot_ts` - Time between first and last detection (in seconds)
- `dist` - Straight line distance between first and last detection site (in metres)
- `rate` - Overall rate of movement (`tot_ts`/`dist`), in metres/second
- `bearing` - Bearing between first and last detection sites
- `num_det` - Number of detections summarized

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Summarize tags
tag_summary <- tagSum(sql_motus)

# For specific SQLite table/view (needs gpsLat/gpsLon) --------------
library(dplyr)
tbl_alltagsGPS <- tbl(sql_motus, "alltagsGPS")
tag_summary <- tagSum(tbl_alltagsGPS)

# For a flattened data frame ----------------------------------------
df_alltagsGPS <- collect(tbl_alltagsGPS)
tag_summary <- tagSum(df_alltagsGPS)

# Can be filtered, e.g., for only a few tags
tag_summary <- tagSum(filter(tbl_alltagsGPS, motusTagID %in% c(16047, 16037, 16039)))
```

`tagSumSite` *Summarize detections of all tags by site*

Description

Creates a summary for each tag of it's first and last detection time at each site, length of time between first and last detection of each site, and total number of detections at each site.

Usage

```
tagSumSite(data, units = "hours")
```

Arguments

`data`	a selected table from .motus data, eg. "alltagsGPS", or a data.frame of detection data including at a minimum variables for motusTagID, fullID, recvDeployName, ts, recvDeployLat, recvDeployLon, gpsLat, gpsLon
`units`	units to display time difference, defaults to "hours", options include "secs", "mins", "hours", "days", "weeks"

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# convert sql file "sql_motus" to a tbl called "tbl_alltags"
library(dplyr)
tbl_alltags <- tbl(sql_motus, "alltagsGPS")

# convert the tbl "tbl_alltags" to a data.frame called "df_alltags"
df_alltags <- tbl_alltags  %>%
  collect() %>%
  as.data.frame()

# Create tag summaries for all tags within detection data with time in
# minutes with tbl file tbl_alltags
tag_site_summary <- tagSumSite(tbl_alltags, units = "mins")

# Create tag summaries for only select tags with time in default hours with
# data.frame df_alltags
tag_site_summary <- tagSumSite(filter(df_alltags,
                                      motusTagID %in% c(16047, 16037, 16039)))

# Create tag summaries for only a select species with data.frame df_alltags
tag_site_summary <- tagSumSite(filter(df_alltags, speciesEN == "Red Knot"))
```

`tellme` *Report how much new data motus has for a tag detection database*

Description

"new" means data not already in your local database.

Usage

```
tellme(projRecv, new = FALSE, dir = getwd())
```

Arguments

`projRecv`	Numeric. Project code from motus.org, *or* character receiver serial number.
`new`	Logical. Create a new database (Default `FALSE`)? Specify `new = TRUE` to create a new local copy of the database to be downloaded. Otherwise, it assumes the database already exists, and will stop with an error if it cannot find it in the current directory. This is mainly to prevent inadvertent downloads of large amounts of data that you already have!
`dir`	Character. Path to the folder where you are storing databases IF `NULL` (default), uses current working directory.

Value

a named list with these items:

- numBatches: number of batches having data for your database
- numRuns: number of runs of tags detections with new data
- numHits: number of new detections
- numGPS: number of new GPS fixes covering the new detections
- numBytes: estimated size of download, in bytes. This is an estimate of the *uncompressed* size, but data are gz-compressed for transfer, so the number of bytes you have to download is typically going to be smaller than this number by a factor of 2 or more.

Note

if you specify `new = TRUE` and the database does not already exist, it will be created (but empty).

`timeToSunriset` *Obtain time to and from sunrise/sunset*

Description

Create and add columns for time to and time since sunrise/sunset to tag data. Can take a motus database table, but will always return a collected data frame. Requires data containing at least latitude, longitude, and time.

Usage

```
timeToSunriset(
  df_src,
  lat = "recvDeployLat",
  lon = "recvDeployLon",
  ts = "ts",
  units = "hours",
  data
)
```

Arguments

`df_src`	Data frame, SQLite connection, or SQLite table. An SQLite connection would be the result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`; an SQLite table would be the result of `dplyr::tbl(tags, "alltags")`; a data frame could be the result of `dplyr::tbl(tags, "alltags") %>% dplyr::collect()`.
`lat`	Character. Name of column with latitude values, defaults to `recvDeployLat`
`lon`	Character. Name of column with longitude values, defaults to `recvDeployLon`
`ts`	Character. Name of column with time as numeric or POSIXct, defaults to `ts`
`units`	Character. Units to display time difference, defaults to "hours", options include "secs", "mins", "hours", "days", "weeks".
`data`	Defunct, use `src`, `df_src`, or `df` instead.

Details

Uses `sunRiseSet()` to perform sunrise/sunset calculates, see `?sunRiseSet` for details regarding how local dates are assessed from UTC timestamps.

Value

Original data (as a flat data frame), with the following additional columns:

- `sunrise` - Time of sunrise in **UTC** for that row's date and location
- `sunset` - Time of sunset in **UTC** for that row's date and location
- `ts_to_set` - Time to next sunset, in `units`

- `ts_since_set` - Time to previous sunset, in `units`
- `ts_to_rise` - Time to next sunrise after, in `units`
- `ts_since_rise` - Time to previous sunrise, in `units`

Examples

```
# Download sample project 176 to .motus database (username/password are "motus.sample")
## Not run: sql_motus <- tagme(176, new = TRUE)

# Or use example data base in memory
sql_motus <- tagmeSample()

# Get sunrise and sunset information for alltags view with units in minutes
sunrise <- timeToSunriset(sql_motus, units = "mins")
```

`writeRunsFilter` *Write to the local database the probabilities associated with runs for a filter*

Description

Write to the local database the probabilities associated with runs for a filter

Usage

```
writeRunsFilter(
  src,
  filterName,
  motusProjID = NA,
  df,
  overwrite = TRUE,
  delete = FALSE
)
```

Arguments

`src`	SQLite connection. Result of `tagme(XXX)` or `DBI::dbConnect(RSQLite::SQLite(), "XXX.motus")`.
`filterName`	Character. Unique name given to the filter
`motusProjID`	Character. Optional project ID attached to the filter in order to share with other users of the same project.
`df`	Data frame. Containing `runID`, `motusTagID` and probability values to save in the local database
`overwrite`	Logical. When `TRUE` ensures that existing records matching the same `filterName` and `runID` get replaced

`delete`	Logical. When TRUE, removes all existing filter records associated with the `filterName` and re-inserts the ones contained in the dataframe. This option should be used if the dataframe provided contains the entire set of filters you want to save.

Value

database connection refering to the filter created

Index