Package ‘timeseriesdb’

August 6, 2018

Type Package
Version 0.4.1
Title Manage Time Series for Official Statistics with R and PostgreSQL
Description Archive and manage time series data from official statistics. The 'timeseriesdb' package was designed to manage a large catalog of time series from official statistics which are typically published on a monthly, quarterly or yearly basis. Thus timeseriesdb is optimized to handle updates caused by data revision as well as elaborate, multi-lingual meta information.

Author ``Matthias Bannert <bannert@kof.ethz.ch> [aut, cre]''
Depends R (>= 3.0.0), RPostgreSQL, jsonlite (>= 1.1), methods
Imports xts, zoo, xtable, shiny, DBI, openxlsx, data.table (>= 1.9.4),
Suggests knitr, testthat
VignetteBuilder knitr
Date 2018-07-19
License GPL-2
URL https://github.com/mbannert/timeseriesdb
BugReports https://github.com/mbannert/timeseriesdb/issues
LazyData true
Maintainer 'Matthias Bannert <bannert@kof.ethz.ch>
RoxygenNote 6.0.1
Encoding UTF-8
NeedsCompilation no
Repository CRAN
Date/Publication 2018-08-06 11:20:03 UTC

R topics documented:

activateTsSet ........................................... 2
addKeysToTsSet ......................................... 3
addMetaInformation ..................................... 4
activateTsSet

Activate a Set of Time Series

Description

Activate a set of time series to get in the user's sight. Deactivated sets are not deleted though.
addKeysToTsSet

Usage

activateTsSet(con, set_name, user_name = Sys.info()$"user",
              tbl = "timeseries_sets", schema = "timeseries")

Arguments

con PostgreSQL connection object
set_name character name of the set to be activated.
user_name character name of the user. Defaults to system user.
tbl character name of set table. Defaults to timeseries\_sets.
schema character name of the database schema. Defaults to timeseries.

Author(s)

Matthias Bannert, Ioan Gabriel Bucur

addKeysToTsSet Add keys to an existing Time Series set

Description

Add keys to an existing Time Series set

Usage

addKeysToTsSet(con, set_name, ts_keys, user_name = Sys.info()$"user",
                tbl = "timeseries_sets", schema = "timeseries")

Arguments

con PostgreSQL connection
set_name The name of the set
ts_keys A character vector of keys to be added
user_name The user name of the set’s owner
tbl Name of the time series sets table
schema Schema of the time series database to use

Author(s)

Severin Thöni
**addMetaInformation**  
*Add Meta Information to R Environments*

**Description**

This function adds meta information to environments that are explicitly meant to store Meta Information. This function can be used separately in interactive R Session or to facilitate mapping database information to R.

**Usage**

```r
addMetaInformation(series, map_list, meta_env = NULL, overwrite_objects = F, overwrite_elements = T)
```

**Arguments**

- `series` character name key of
- `map_list` list to represent key value mapping. Could also be of class miro.
- `meta_env` an environment that already holds meta information and should be extended. Defaults to NULL in which case it creates and returns a new environment.
- `overwrite_objects` logical should the entire existing meta information be overwritten inside the environment? Defaults to FALSE
- `overwrite_elements` logical should single matching elements of a meta information objectes be overwritten. Defaults to TRUE.

**beginTransaction**  
*Convenience Wrapper to SQL classics for BEGIN, COMMIT, ROLLBACK*

**Description**

this set of function can speed up loops by starting a transaction, performing several queries and ending them with either commit or rollback.

**Usage**

```r
beginTransaction(con, quiet = T)
commitTransaction(con, quiet = T)
rollbackTransaction(con, quiet = T)
```
**changeTsSetOwner**

**Arguments**

- **con**: PostgreSQL connection object.
- **quiet**: logical should the query be executed quietly? Otherwise BEGIN, COMMIT or ROLLBACK are echoed.

**Description**

Change the owner of a Time Series set

**Usage**

```r
callchangeTsSetOwner(con, set_name, old_owner = Sys.info("user"), new_owner, 
                    tbl = "timeseries_sets", schema = "timeseries")
```

**Arguments**

- **con**: PostgreSQL connection
- **set_name**: Name of the set to be updates
- **old_owner**: User name of the set’s current owner
- **new_owner**: User name of the set’s new owner
- **tbl**: Name of the time series sets table
- **schema**: Schema of the time series database to use

**Author(s)**

Severin Thöni

---

**createConObj**

**Conveniently Create Connection Object to PostgreSQL based timeseriesdb**

**Description**

Create a connection object while getting user information from the R session. Also standard db parameters like port and driver are set. Yet flexible information like host or dbname should be added to Sys.setenv environments.

**Usage**

```r
callcreateConObj(dbuser = Sys.info("user"), 
                dbname = Sys.getenv("TIMESERIESDB_NAME"), 
                dbhost = Sys.getenv("TIMESERIESDB_HOST"), passwd, dbport = 5432)
```
createHstore

**Arguments**

- **dbuser**: character username. Defaults to reading username from Sys.info().
- **dbname**: character name of the database, assumes dbname is stored in TIMESERIESDB_NAME.
- **dbhost**: character host address, assumes dbhost is stored in TIMESERIESDB_HOST.
- **passwd**: character password is used. No defaults, best way to pass a password is to use .rs.askForPassword to hide password entries when using R Studio.
- **dbport**: integer port number defaults to 5432 for postgres

**createHstore**

**Create Hstore**

**Description**

Function to Create Hstore Key Value Pair Mapping

**Usage**

createHstore(x, ...)

```r
## S3 method for class 'ts'
createHstore(x, ...)

## S3 method for class 'zoo'
createHstore(x, ...)

## S3 method for class 'data.frame'
createHstore(x, ...)

## S3 method for class 'list'
createHstore(x, ...)
```

**Arguments**

- **x**: a time series object, a two column data frame or object of S3 class miro (meta information for R objects).
- **...**: optional arguments, fct = TRUE create text expressions of hstore function calls.
  also for data.frames key_pos and value_pos could be given if they are different from 1 and 2. e.g. position of the key col and position of the value col in a data.frame.

**Details**

This function creates a key value pair mapping from a time series object. It returns an hstore object that can be inserted to a PostgreSQL database relation field of type hstore.
**createTimeseriesMain**

**Author(s)**
Matthias Bannert

**Examples**
```r
tsl <- ts(rnorm(100), start = c(1990,1), frequency = 4)
createHstore(tsl)
```

**Description**
These functions create statements to set up 5 Tables used to manage and archive time series information in PostgreSQL. Make sure you have sufficient rights to create relations in your PostgreSQL schema. These functions are only used for an initial setup. You can either run this group of functions separately or use `runCreateTables` to run all functions at once.

**Usage**
```r
createTimeseriesMain(schema = "timeseries", tbl = "timeseries_main")
createTimeseriesVintages(schema = "timeseries", tbl = "timeseries_vintages")
createTimeseriesSets(schema = "timeseries", tbl = "timeseries_sets")
createMetaUnlocalized(schema = "timeseries", tbl = "meta_data_unlocalized", main = "timeseries_main")
createMetaLocalized(schema = "timeseries", tbl = "meta_data_localized", main = "timeseries_main")
createMetaDatasets(schema = "timeseries", tbl = "meta_datasets")
```

**Arguments**
- `schema` character denoting a PostgreSQL schema
- `tbl` character denoting a table name
- `main` character denoting name of the main table for referencing. This argument is only available to meta data statements.
Details

The following tables will be create in the given schema.

- "timeseries_main" contains time series themselves as hstore key value pairs.
- "timeseries_vintages" contains vintages of time series. This is useful for published data that can be revised. See also OECD definition of vintages.
- "timeseries_sets" contains a vector of time series keys. This table can be used like a shopping cart in an e-commerce application.
- "meta_data_unlocalized" contains translation agnostic meta information, e.g., username.
- "meta_data_localized" contains translation specific meta information, e.g., wording of a question.

References


dbIsValid, PostgreSQLConnection-method

Check Validity of a PostgreSQL connection

Description

Is the PostgreSQL connection expired?

Usage

```r
## S4 method for signature 'PostgreSQLConnection'
dbIsValid(dbObj)
```

Arguments

dbObj PostgreSQL connection object.

deactivateTsSet, Deactivate a Set of Time Series

Description

This deactivates a set of time series to get out of the user’s sight, but it’s not the deleted because users may not delete sets.

Usage

```r
deactivateTsSet(con, set_name, user_name = Sys.info()$"user",
               tbl = "timeseries_sets", schema = "timeseries")
```
**deleteTimeSeries**

**Arguments**

- **con** PostgreSQL connection object
- **set_name** character name of the set to be deactivated.
- **user_name** character name of the user. Defaults to system user.
- **tbl** character name of set table. Defaults to timeseries\_sets.
- **schema** character name of the database schema. Defaults to timeseries.

**Author(s)**

Matthias Bannert, Ioan Gabriel Bucur

---

**deleteTimeSeries**  
*Delete Time Series from the database*

**Description**

This function deletes time series AND their metainformation from the database. All meta information in all series will be deleted. To only edit the original time series use `storeTimeSeries` to overwrite existing series.

**Usage**

```python
deleterTimeSeries(series, con, chunksize = 10000,
                  tbl_main = "timeseries_main", schema = "timeseries")
```

**Arguments**

- **series** character name of the timeseries
- **con** a PostgreSQL connection object
- **chunksize** integer max size of chunk when deleting chunkwise. Defaults to 10000.
- **tbl_main** character name of the table that contains the main time series catalog. Defaults to 'timeseries\_main'.
- **schema** SQL schema name. Defaults to 'timeseries'.
deleteTsSet  
*Permanently delete a Set of Time Series Keys*

**Description**

Permanently delete a Set of Time Series Keys

**Usage**

```r
deleteTsSet(con, set_name, user_name = Sys.info()[$"user"],
             tbl = "timeseries_sets", schema = "timeseries")
```

**Arguments**

- `con`  
  PostgreSQL connection object
- `set_name`  
  The name of the set to be deleted
- `user_name`  
  Username to which the set belongs
- `tbl`  
  Name of set table
- `schema`  
  Name of timeseries schema

**Author(s)**

Severin Thöni

---

exportMetaEnv  
*Export Content of Meta Information Environment to Various File Formats*

**Description**

The idea of this function is to create a standalone meta information catalog. The catalog file can be used as a companion to illustrate time series exports from timeseriesdb. Note that this function imports functionality from other packages such as data.table and openxlsx.

**Usage**

```r
exportMetaEnv(meta_env, fname = NULL, export_type = "pdf",
              flexcols = NULL, row.names = F, sep = ";", overwrite = T)
```
getListDepth

Arguments

- **meta_env**: meta\_env environment object.
- **fname**: character file name including file extension. If set to NULL no file is export. The resulting data.frame is just displayed on the console in this case. Defaults to NULL.
- **export_type**: character indication which file format should be exported. "pdf", "tex", "csv" are the eligible.
- **flexcols**: which columns should be kept in the data.frame. Defaults to NULL, using all columns.
- **row.names**: logical should row.names be displayed in csv.
- **sep**: character separator
- **overwrite**: should existing files be overwritten? Defaults to TRUE.

getListDepth

*Determine depth of a list*

Description

This function recursively checks the depth of a list and returns an integer value of depth

Usage

getListDepth(this)

Arguments

- **this**: an object of class list

Details

Hat tip to flodel at stackoverflow for suggesting this light weight way analyze depth of a nested list. Further complexity needs to be added to cover the fact that data.frame are lists, too. A more sophisticated recursive function can be found in the gatveys2 package.

References

**getMeta**  
*Quickly Handle Meta Information*

**Description**

Sometimes reading the entire meta description for all language or multiple time series might not be necessary. Quick handle operators help users to access the information quickly as a non-nested list for only one language is returned. These functions are alpha status, more will follow.

**Usage**

getMeta(series, lang, con, tbl = "meta_data_localized",  
schema = "timeseries")

**Arguments**

- `series`: an R time series object
- `lang`: character name of the language of the meta information. Typically 'de', 'it', 'fr' or 'en'.
- `con`: connection object
- `tbl`: character name of the table that contains the meta information.
- `schema`: SQL schema name. Defaults to 'timeseries'.

**getTimeSeriesVintages**  
*Get all available vintages for the time series identified by series*

**Description**

Get all available vintages for the time series identified by `series`

**Usage**

getTimeSeriesVintages(series, con, tbl_vintages = "timeseries_vintages",  
schema = "timeseries")

**Arguments**

- `series`: character Names of the time series for which to get the vintages
- `con`: PostgreSQL connection object.
- `tbl_vintages`: character string denoting the name of the vintages time series table in the PostgreSQL database.
- `schema`: SQL schema name. Defaults to timeseries.
indexToDate

Convert ts style time index Date representation

Description

Helper function to convert time series indices of the form 2005.75 to a date representation like 2005-07-01. Does not currently support sub-monthly frequencies.

Usage

indexToDate(x, as.string = FALSE)

Arguments

x numeric A vector of time series time indices (e.g. from stats::time)

as.string logical If as.string is TRUE the string representation of the Date is returned, otherwise a Date object.

Author(s)

Severin Thöni

joinTsSets

Join two Time Series sets together

Description

This will create a new set set_name_new with the keys from both set_name_1 and set_name_2 combined. By default the description will be a combination of the descriptions of the subsets and the new set will only be active if BOTH subsets were active.

Usage

joinTsSets(con, set_name_1, set_name_2, set_name_new,
user_name1 = Sys.info()$["user"], user_name2 = user_name1,
user_name_new = user_name1, description = NULL, active = NULL,
tbl = "timeseries_sets", schema = "timeseries")
**listTsSets**

*List All Time Series Sets for a Particular User*

**Arguments**

- **con**: PostgreSQL connection
- **set_name_1**: Name of the first set
- **set_name_2**: Name of the second set
- **set_name_new**: Name of the set to be created
- **user_name1**: User name of the first set’s owner
- **user_name2**: User name of the second set’s owner
- **user_name_new**: User name of the new set’s owner
- **description**: Description of the new set
- **active**: Should the new set be marked as active
- **tbl**: The time series set table
- **schema**: The time series db schema to use

**Author(s)**

Severin Thöni

**Description**

Show the names of all sets that are available to a particular user.

**Usage**

```python
listTsSets(con, user_name = Sys.info()"user", tbl = "timeseries_sets", 
           schema = "timeseries")
```

**Arguments**

- **con**: PostgreSQL connection object
- **user_name**: character name of the user. Defaults to system user.
- **tbl**: character name of set table. Defaults to timeseries\_sets.
- **schema**: character name of the database schema. Defaults to timeseries.

**Author(s)**

Matthias Bannert, Gabriel Bucur
**loadTsSet**

*Load a Time Series Set*

**Description**

Loads a Time Series Set.

**Usage**

```python
loadTsSet(con, set_name, user_name = Sys.info()["user"],
           tbl = "timeseries_sets", schema = "timeseries")
```

**Arguments**

- **con**: PostgreSQL connection object
- **set_name**: character name of the set to be loaded.
- **user_name**: character name of the user. Defaults to system user.
- **tbl**: character name of set table. Defaults to timeseries\_sets.
- **schema**: character name of the database schema. Defaults to timeseries.

**Author(s)**

Matthias Bannert, Ioan Gabriel Bucur

---

**overwriteTsSet**

*Overwrite a Time Series Set with a new one*

**Description**

Completely replaces the set set\_name of user\_name with the new values (keys, description, active) of the new one. If the set does not yet exist for the given user it will be created.

**Usage**

```python
overwriteTsSet(con, set_name, ts_keys, user_name = Sys.info()["user"],
               description = ", description = "", active = TRUE, tbl = "timeseries_sets",
               schema = "timeseries")
```

---
Arguments

- **con**: PostgreSQL connection
- **set_name**: The name of the set to be overwritten
- **ts_keys**: The keys in the new set
- **user_name**: The owner of the set to be overwritten
- **description**: The description of the new set
- **active**: Should the new set be active?
- **tbl**: Name of the time series sets table
- **schema**: Schema of the time series database to use

Author(s)

Severin Thöni

---

```
pgCopyDf

pgCopyDf

Description

Copy data.frame to postgres using bulk copy

Usage

pgCopyDf(con, d, q, chunksize = 10000)

Arguments

- **con**: PostgreSQL connection object.
- **d**: data.frame
- **q**: character string containing a SQL query.
- **chunksize**: integer, defaults to 10000.
```
readMetaInformation  

Read Meta Information from a Time Series Database

Description

This function reads meta information from a timeseriesdb package PostgreSQL database and puts into a meta information environment.

Usage

readMetaInformation(series, con, locale = "de", tbl = "meta_data_localized", overwrite_objects = F, overwrite_elements = T, meta_env = NULL, schema = "timeseries")

Arguments

series  
character name of a time series object.

con  
PostgreSQL connection object

locale  
character denoting the locale of the meta information that is queried. defaults to 'de' for German. At the KOF Swiss Economic Institute meta information should be available als in English 'en', French 'fr' and Italian 'it'. Set the locale to NULL to query unlocalized meta information.

tbl  
character name of the table that contains meta information. Defaults to 'meta_data_localized'. Choose meta 'meta_data_unlocalized' when locale is set to NULL.

overwrite_objects  
logical should the entire object for a key be overwritten. Defaults to FALSE.

overwrite_elements  
logical should single elements inside the environment be overwritten. Defaults to TRUE.

meta_env  
environment to which the meta information should be added. Defaults to NULL. In this case an environment will be returned. If you run this function in a loop best create an empty environment before the loop or apply call and pass the environment to this function. By doing so new elements will be added to the environment.

schema  
SQL schema name. Defaults to timeseries.
readTimeSeries

Description

This function reads a time series from a PostgreSQL relation that uses Postgres’ key value pair storage (hstore). After reading the information from the database a standard R time series object of class 'ts' is built and returned. Irregular time series return zoo objects.

Usage

readTimeSeries(series, con, valid_on = NULL, tbl = "timeseries_main", tbl_vintages = "timeseries_vintages", schema = "timeseries", env = NULL, pkg_for_irreg = "xts", chunksize = 10000, respect_release_date = FALSE, regex = FALSE)

Arguments

- **series**: character vector of time series keys
- **con**: a PostgreSQL connection object
- **valid_on**: character date string on which the series should be valid. Defaults to NULL. Only needed when different vintages of a time series are stored.
- **tbl**: character string denoting the name of the relation that contains ts_key, ts_data, ts_frequency.
- **tbl_vintages**: character table name of the relation that holds time series vintages
- **schema**: character SQL schema name. Defaults to timeseries.
- **env**: environment, optional argument to dump time series directly into an environment. Most often used with globalenv(), which gives all time series directly back to the global env.
- **pkg_for_irreg**: character name of package for irregular series. xts or zoo, defaults to xts.
- **chunksize**: numeric value of threshold at which input vector should be processed in chunks. defaults to 70000.
- **respect_release_date**: logical should the release set in the database be respected. If TRUE, the last observation will be cut off if server time is before release date. Reasonable for release date.
- **regex**: If set to TRUE, series will be interpreted as a regular expression, so that all time series whose keys match the pattern will be returned.

Author(s)

Matthias Bannert, Gabriel Bucur
 Remove keys from a Time Series set (if present)

Description

Remove keys from a Time Series set (if present)

Usage

```r
removeKeysFromTsSet(con, set_name, ts_keys, user_name = Sys.info()["user"],
                   tbl = "timeseries_sets", schema = "timeseries")
```

Arguments

- `con`: PostgreSQL connection
- `set_name`: character name of a time series set.
- `ts_keys`: A character vector of keys to be removed.
- `user_name`: The user name of the set's owner.
- `tbl`: Name of the time series sets table.
- `schema`: Schema of the time series database to use.

Author(s)

Severin Thöni

---

Delete all objects except for specific objects

Description

Run `rm(list=ls())` but sparing some objects from being deleted. This function is particularly handy when you want to clear the memory but want to keep the database connection object.

Usage

```r
rmAllBut(but, env = .GlobalEnv, quiet = FALSE)
```

Arguments

- `but`: character vector of variables that should not be deleted.
- `env`: environment to clean up. Defaults to `.GlobalEnv`
- `quiet`: logical should functions print output? Defaults to `FALSE`.
runCreateTables  
*Run Setup: Create all mandatory tables*

**Description**

Creates all tables absolutely needed for timeseriesdb to work correctly. This function should only be run once as an initial setup. Make sure you got sufficient access rights. The function returns a list of status reports for the its 5 database queries. look at this helps you to see whether anything went wrong.

**Usage**

```r
runCreateTables(con, schema = "timeseries")
```

**Arguments**

- `con`  
  PostgreSQL connection object. Typically created with `createConObj`.
- `schema`  
  character denoting a PostgreSQL schema.

runDbQuery  
*Run SELECT query*

**Description**

Run database queries using `dbSendQuery`, `fetch` and `dbClearResult` in similar fashion as `dbGetQuery` but provide better error handling. This function always returns a data.frame as opposed to different types in case of an exception. However, if the database query fails and empty data.frame is returned. Besides query status and database error are returned as attributes. Make sure to use BEGIN and COMMIT outside of these statements.

**Usage**

```r
runDbQuery(con, sql_query, ...)
```

**Arguments**

- `con`  
  PostgreSQL connection object
- `sql_query`  
  character string containing a SQL query
- `...`  
  Additional arguments to be passed to `dbGetQuery`

**Examples**

```r
# There's no connection, so this returns a proper error message.

out_obj <- runDbQuery(bogus_connection,"SELECT * FROM some_table")
attributes(out_obj)
```
**runUpgradeTables**

Add Release Date Column to Tables

**Description**

Adds a release column to tables of older versions of timeseriesdb.

**Usage**

```python
runUpgradeTables(con, schema = "timeseries")
```

**Arguments**

- **con** PostgreSQL connection object
- **schema** database schema, defaults to 'timeseries'.

---

**searchKVP**

Search Key-Value Pairs, look for existing keys in an Hstore

**Description**

Search hstore key value in PostgreSQL. Very handsome when crawling the database by meta information. Currently works for non translated meta information.

**Usage**

```python
searchKVP(key, value, con = getSys.getenv("TIMESERIESDB_CON")),
    hstore = "meta_data", tbl = "meta_data_unlocalized", where = NULL,
    schema = "timeseries")
```

```python
lookForKey(key, con = getSys.getenv("TIMESERIESDB_CON")),
    hstore = "meta_data", tbl = "meta_data_unlocalized", where = NULL,
    schema = "timeseries")
```

**Arguments**

- **key** character in the hstore
- **value**
- **con** PostgreSQL connection object
- **hstore** name of the hstore column
- **tbl** name of the table to be queried. defaults to 'meta_dataLocalized'
- **where** character restrict the SQL query by an additional where clause. Defaults to NULL.
- **schema** SQL schema name. defaults to timeseries. E.g.: ts_key LIKE ...
setAttrListWise  
**Set Attributes to Each Element of List According to a Given Vector**

**Description**

An attribute is set to all elements of a list given a vector of possible instances of the attribute. Note that this function fails to execute if the vector is not of the same length list.

**Usage**

```r
setAttrListWise(li, attrib, vec)
```

**Arguments**

- `li` a list
- `attrib` character name of the attribute
- `vec` vector containing all instances of the attribute

storeListChunkWise  
**Store a List of Time Series Chunk Wise to Avoid Memory Problem**

**Description**

This function is a wrapper around `storeTimeSeries`. It is used to split large lists of time series according to memory limitations. This function uses INSERT INTO instead of the more convenient `dbWritetable` for performance reasons. DO NOT USE THIS FUNCTIONS IN LOOPS OR LAPPLY! This function can handle a set of time series on its own and is much faster than looping over a list. Non-unique primary keys are overwritten!

**Usage**

```r
storeListChunkWise(series, con, li = NULL, tbl = "timeseries_main", 
                   md_unlocal = "meta_data_unlocalized", overwrite = T, chunksize = 10000, 
                   schema = "timeseries", show_progress = FALSE)
```

**Arguments**

- `series` character name of a time series, S3 class ts. When used with lists it is convenient to set series to names(li). Note that the series name needs to be unique in the database!
- `con` a PostgreSQL connection object.
- `li` list of time series. Defaults to NULL to no break legacy calls that use lookup environments.
storeMetaChunkWise

```r
storeMetaChunkWise(meta_envir, con, schema = "timeseries",
                    tbl = "meta_data_unlocalized", keys = NULL, chunksize = NULL,
                    quiet = T)
```

**Arguments**

- `meta_envir` object of class `meta_env`. Most likely generated by `addMetaInformation`
- `con` a PostgreSQL connection object
- `schema` character name of the schema to write to. Defaults to ‘timeseries’.
- `tbl` character name of the meta information table to write to. Defaults to ‘meta_data_unlocalized’.
- `keys` character vector of time series. If specified only the selected meta information is stored. Defaults to NULL which stores all meta information records in the environment.
- `chunksize` integer number of chunks. Defaults to NULL which automatically choose chunks based on Cstack size.
- `quiet` logical should the update function be quiet? Defaults to TRUE.

**Description**

FUNCTION DEPRECATED. This function is a wrapper around `updateMetaInformation`. It is used to split large environments of meta info to avoid memory limitations. This function uses INSERT INTO instead of the more convenient dbWritetable for performance reasons. DO NOT USE THIS FUNCTIONS IN LOOPS OR LAPPLY! This function can handle a set of time series on its own and is much faster than looping over a list. Non-unique primary keys are overwritten!

**Author(s)**

Matthias Bannert, Gabriel Bucur
storeMetaInformation  

*Store Meta Information to the Database*

**Description**

This function stores meta information to the database for a given time series. Make sure that corresponding time series had been inserted to the main table before.

**Usage**

```r
storeMetaInformation(series, con, tbl = "meta_data_localized",
                     lookup_env = "meta_data_localized", locale = "de",
                     overwrite = FALSE, quiet = FALSE, schema = "timeseries")
```

**Arguments**

- `series` a character name of an time series object
- `con` a PostgreSQL connection object
- `tbl` name of the meta information table, defaults to localized meta data: `meta_data_localized`. Alternatively choose `meta_data_unlocalized` if you are not translating meta information.
- `lookup_env` name of the R environment in which to look for meta information objects
- `locale` character locale for the metainformation. Defaults to German 'de'. See also `readMetaInformation`. If locale is set to NULL unlocalized meta is updated. Make sure to change tbl to `meta_data_unlocalized`.
- `overwrite` logical, defaults to FALSE.
- `quiet` logical, should there be console output for every query result? Defaults to FALSE.
- `schema` SQL schema name, defaults to 'timeseries'.

storeTimeSeries  

*Write an R time series to a PostgreSQL database*

**Description**

This function writes time series object into a relational PostgreSQL database make use of PostgreSQL own 'key' => 'value' storage called `hstore`. The schema and database needs to created first. The parent R Package of this functions suggests a database structure designed to store a larger amount of time series. This function uses `INSERT INTO` instead of the more convenient `dbWriteTable` for performance reasons. DO NOT USE THIS FUNCTIONS IN LOOPS OR LAPPLY! This function can handle a set of time series on its own and is much faster than looping over a list. Non-unique primary keys are overwritten!
Usage

storeTimeSeries(series, con, li = NULL, valid_from = NULL,
release_date = NULL, store_freq = T, tbl = "timeseries_main",
tbl_vintages = "timeseries_vintages",
md_unlocal = "meta_data_unlocalized", lookup_env = .GlobalEnv,
overwrite = T, schema = "timeseries")

Arguments

series character name of a time series, S3 class ts. When used with lists it is convenient
to set series to names(li). Note that the series name needs to be unique in the
database!

con a PostgreSQL connection object.

li list of time series. Defaults to NULL to no break legacy calls that use lookup
environments.

valid_from character date lower bound of a date range.

release_date character date string indicating when a series should be released. This facilitates
implementations that only share part of the information before a certain release
date.

store_freq logical, should frequencies be stored. Defaults to TRUE.

tbl character string denoting the name of the main time series table in the Post-
greSQL database.

tbl_vintages character string denoting the name of the vintages time series table in the Post-
greSQL database.

md_unlocal character string denoting the name of the table that holds unlocalized meta in-
formation.

lookup_env environment to look in for timeseries. Defaults to .GlobalEnv.

overwrite logical should existing records (same primary key) be overwritten? Defaults to
TRUE.

schema SQL schema name. Defaults to timeseries.

Author(s)

Matthias Bannert, Charles Clavadetscher, Gabriel Bucur

Description

Store a new set of Time Series to the database. Users can select the time series keys that should be
grouped inside a set.
Usage

storeTsSet(con, set_name, set_keys, user_name = Sys.info()["user"],
description = ",", active = TRUE, tbl = "timeseries_sets",
schema = "timeseries"")

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>con</td>
<td>PostgreSQL connection object</td>
</tr>
<tr>
<td>set_name</td>
<td>character name of a set time series in the database.</td>
</tr>
<tr>
<td>set_keys</td>
<td>list of keys contained in the set and their type of key.</td>
</tr>
<tr>
<td>user_name</td>
<td>character name of the user. Defaults to system user.</td>
</tr>
<tr>
<td>description</td>
<td>character description of the set to be stored in the db.</td>
</tr>
<tr>
<td>active</td>
<td>logical should a set be active? Defaults to TRUE. If set to FALSE a set is not seen directly in the GUI directly after being stored and needs to be activated first.</td>
</tr>
<tr>
<td>tbl</td>
<td>character name of set table. Defaults to timeseries_sets.</td>
</tr>
<tr>
<td>schema</td>
<td>character name of the database schema. Defaults to timeseries.</td>
</tr>
</tbody>
</table>

Author(s)

Ioan Gabriel Bucur, Matthias Bannert, Severin Thöni

---

**updateMetaInformation**  
*Update Meta Information Records*

**Description**

When a time series is stored to the database by `storeTimeSeries` a minimal unlocalized (i.e. untranslated) meta information record is being generated. This meta information can be supplement using the `updateMetaInformation` methods. Depending on the class of the environment that holds the meta information localized or unlocalized meta information is updated. NOTE: AVOID looping over this function. This functions accepts entire environments and creates large SQL queries instead of looping over multiple small queries. In other words loops are moved to the database level for massive speed gain.

**Usage**

updateMetaInformation(meta, con, schema = "timeseries",
tbl = "meta_data_unlocalized", locale = NULL, keys = NULL, quiet = F,
chunksize = 10000)

```r
## S3 method for class 'meta_env'
updateMetaInformation(meta, con, schema = "timeseries",
tbl = "meta_data_unlocalized", locale = NULL, keys = NULL, quiet = F,
chunksize = 10000)
```
**writeLogFile**

**Simple Log File Writer**

**Description**

Most simple log file writer just write steps of a script to a text file.

**Usage**

```r
writeLogFile(msg, filename = NULL, line_end = "\n")
```

**Arguments**

- `msg`: log file message
- `filename`: character name of a textfile. Defaults to NULL.
- `line_end`: line end character

**zooLikeDateConvert**

**Zoo like Date Conversion**

**Description**

This function is taken from the zoo package. It is basically the S3 method as.Date.numeric of the package zoo. It is used to turn 2005.75 (3rd quarter of 2005) like date formats into dates like 2005-07-01.

**Usage**

```r
zooLikeDateConvert(x, offset = 0, ...)
```
Arguments

x object of class ts or zoo (experimental)
offset numeric defaults to 0. See the zoo package for more information.
... optional arguments.

Author(s)

Achim Zeileis, Gabor Grothendieck, Jeffrey A. Ryan, Felix Andrews