Package 'arealDB'

July 1, 2020

Type Package

Title Harmonise and Integrate Heterogeneous Areal Data

Description Many relevant applications in the environmental and socioeconomic sciences use areal data, such as biodiversity checklists, agricultural statistics, or socioeconomic surveys. For applications that surpass the spatial, temporal or thematic scope of any single data source, data must be integrated from several heterogeneous sources. Inconsistent concepts, definitions, or messy data tables make this a tedious and error-prone process. 'arealDB' tackles those problems and helps the user to integrate a harmonised databases of areal data. Read the pre-print at Ehrmann, Seppelt & Meyer (2020) <arXiv:1909.06610>.

Version 0.3.4

URL https://gitlab.com/luckinet/software/arealDB

BugReports https://gitlab.com/luckinet/software/arealDB/issues

Depends R (>= 2.10)

Language en-gb

License GPL-3

Encoding UTF-8

LazyData true

Imports checkmate, dplyr, magrittr, readr, rlang, sf, stringr, tibble, tidyr, tidyselect, tabshiftr

Suggests testthat, knitr, rmarkdown, covr

RoxygenNote 7.1.0

NeedsCompilation no

Author Steffen Ehrmann [aut, cre] (<https://orcid.org/0000-0002-2958-0796>), Arne Rümmler [aut, ctb] (<https://orcid.org/0000-0001-8637-9071>), Felipe Melges [ctb] (<https://orcid.org/0000-0003-0833-8973>), Carsten Mayer [ctb] (<https://orcid.org/0000-0003-3927-5856>)

Maintainer Steffen Ehrmann <steffen.science@funroll-loops.de>

Repository CRAN

Date/Publication 2020-07-01 09:10:06 UTC

R topics documented:

countries . . . 2
getColTypes . . . 3
makeExampleDB . . . 3
matchUnits . . . 4
matchVars . . . 5
normGeometry . . . 5
normTable . . . 8
regDataseries . . . 10
regGeometry . . . 11
regTable . . . 13
setPath . . . 16
setVariables . . . 17
testCompressed . . . 18
translateTerms . . . 19
updateTable . . . 20
View_sf . . . 20

Index **21**

`countries` *Simple feature geometries of countries*

Description

Simple feature geometries of countries

Usage

```
countries
```

Format

The object of class sf has 10 columns and 248 rows. It lists for each country the `unit` name, the `nation` name, `iso_a2` and `iso-a3`-code, whether the country is an `un_member` or an `island` on which `continent` the country is and of which `region` and `subregion` that is part. Finally, it lists the initial ahID for each nation.

getColTypes *Get the column types of a tibble*

Description

Get the column types of a tibble

Usage

```
getColTypes(input = NULL)
```

Arguments

input	[tibble(1)] tibble from which to get column types.

makeExampleDB *Build an example database*

Description

This function helps setting up an example database up until a certain point.

Usage

```
makeExampleDB(until = NULL, path = NULL, verbose = FALSE)
```

Arguments

until	[character(1)] The database building step in terms of the function names until which the example database shall be built, one of "setPath", "setVariables", "regDataseries", "regGeometry", "regTable", "normGeometry" or "normTable".
path	[character(1)] The database gets created by default in tempdir(), but if you want it in a particular location, specify that in this argument.
verbose	[logical(1)] be verbose about building the example database (default FALSE).

Value

No return value, called for the side effect of creating an example database at the specified path.

Examples

```
# to merely register a set of files
makeExampleDB(until = "regTable")

# to build the full example database
makeExampleDB()
```

`matchUnits` *Determine the administrative hierarchy ID*

Description

This function matches territorial units with a list of known units to derive the administrative hierarchy ID.

Usage

```
matchUnits(input = NULL, source = NULL, verbose = FALSE)
```

Arguments

`input`	[`data.frame(1)`] table in which to match administrative units.
`source`	[`integerish(1)`] the geometry ID (`geoID`) from which the terms have been taken.
`verbose`	[`logical(1)`] be verbose about what is happening (default `FALSE`).

Details

`names(input)` must contain at least the `al1 = ...` to match at least a certain nation. Further administrative levels are denoted by column names `al2`, `al3`, ...

Value

The table provided in `input`, where the given columns are replaced by the column `ahID`, which contains the administrative hierarchy ID.

matchVars *Determine the valid ID of variables*

Description

This function matches the values of a variable with an index and returns the specified IDs.

Usage

```
matchVars(input = NULL, source = NULL, ...)
```

Arguments

input	[character(.)] terms to be translated.
source	[integerish(1)] the census ID (cenID) from which the terms have been taken.
...	[list(1)] lists that capture the variables by which to match and the new column names containing the resulting ID; see Details.

Details

Arguments in ... are named lists that indicate with which target column variables shall be matched and which value should be used as target ID.

targetID = list(variable = targetColumn)

The variable must be present as column in input and a table that is named "id_VARIABLE.csv" must be available in the root directory of the project. This should have been created with setVariables.

Value

The table provided in input, where the given variable is replaced by the column that is specified by the argument name.

normGeometry *Normalise geometries*

Description

Harmonise and integrate geometries in a standardised format

Usage

```
normGeometry(
  input = NULL,
  ...,
  thresh = 10,
  outType = "gpkg",
  pattern = NULL,
  update = FALSE,
  verbose = FALSE
)
```

Arguments

input	[character(1)] path of the file to normalise. If this is left empty, all files at stage two as subset by pattern are chosen.
...	[character(.)] a subset of administrative units as given by nation, continent, region, subregion or un_member = TRUE/FALSE. Valid selection criteria and their values are in the object countries.
thresh	[integerish(1)] the deviation of percentage of overlap above which to consider two territorial units "different".
outType	[character(1)] the output file-type, see st_drivers for a list. If a file-type supports layers, they are stored in the same file, otherwise the different layers are provided separately.
pattern	[character(1)] an optional regular expression. Only dataset names which match the regular expression will be returned.
update	[logical(1)] whether or not the physical files should be updated (TRUE) or the function should merely return the geometry inventory of the handled files (FALSE, default). This is helpful to check whether the metadata specification and the provided file(s) are properly specified.
verbose	[logical(1)] be verbose about what is happening (default FALSE). Furthermore, you can use suppressMessages to make this function completely silent.

Details

To normalise geometries, this function proceeds as follows:

1. Read in input and extract initial metadata from the file name.
2. Loop through every nation potentially included in the file that shall be processed and carry out the following steps:
   - In case the geometries are provided as a list of simple feature POLYGONS, they are dissolved into a single MULTIPOLYGON per nation.

- In case the nation to which a geometry belongs has not yet been created at stage three, the following steps are carried out:
  (a) Check whether the recent dataset is GADM, to build the initial administrative hierarchy from GADM geometries, and stop if this is not the case.
  (b) Extract the full hierarchy of all territorial units that are part of the geometry.
  (c) Reconstruct ahID from the intermediate spatial objects and from the metadata.
- In case the nation to which the geometry belongs has already been created, the following steps are carried out:
  (a) Check whether the new geometries have the same coordinate reference system as the already existing database and re-project the new geometries if this is not the case.
  (b) Check whether all new geometries are already exactly matched spatially and stop if that is the case.
  (c) Check whether the new geometries are all within the already defined parents, and save those that are not as a new geometry.
  (d) Calculate spatial overlap and distinguish the geometries into those that overlap with more and those with less than `thresh`.
  (e) For all units that did match, copy ahID from the geometries they overlap.
  (f) For all units that did not match, rebuild metadata and a new ahID.
- If update = TRUE, store the processed geometry at stage three.

3. Move the geometry to the folder '/processed', if it is fully processed.

Value

This function harmonises and integrates so far unprocessed geometries at stage two into stage three of the geospatial database. It produces for each nation in the registered geometries a spatial file of the specified file-type.

See Also

Other normalisers: `normTable()`

Examples

```
library(sf)

# build the example database
makeExampleDB(until = "regGeometry")

# normalise all geometries ...
normGeometry(nation = "estonia", update = TRUE)

# ... and check the result
output <- st_read(paste0(tempdir(), "/newDB/adb_geometries/stage3/estonia.gpkg"))
```

normTable *Normalise data tables*

Description

Harmonise and integrate data tables into standardised format

Usage

```
normTable(
  input = NULL,
  ...,
  source = "tabID",
  pattern = NULL,
  update = FALSE,
  keepOrig = FALSE,
  verbose = FALSE
)
```

Arguments

input	[character(1)] path of the file to normalise. If this is left empty, all files at stage two as subset by pattern are chosen
...	[list(.)] matching lists that capture the variables by which to match and the new column names containing the resulting ID; see Details.
source	[charcter(1)] the source from which translations of terms should be sought. By default the recent "tabID", but when the same terms occur in several tables of a dataseries, chose "datID".
pattern	[character(1)] an optional regular expression. Only dataset names which match the regular expression will be returned.
update	[logical(1)] whether or not the physical files should be updated (TRUE) or the function should merely return the new object (FALSE, default). This is helpful to check whether the metadata specification and the provided file(s) (translation and ID tables) are properly specified.
keepOrig	[logical(1)] to keep the original units and variable names in the output (TRUE) or to remove them (FALSE, default). Useful for debugging.
verbose	[logical(1)] be verbose about translating terms (default FALSE). Furthermore, you can use suppressMessages to make this function completely silent.

Details

Arguments in ... are so-called matching lists. This argument captures three kinds of information:

1. the 'variable' that should be matched with a matching list,
2. the 'targetColumn' in that matching list that should be included in the final table in the place of 'variable' and
3. the 'targetID' (column name) of that new variable.

targetID = list(variable = targetColumn)

'variable' must be present as column in `input` and a table that is named "id_variable.csv" (where 'variable' is replaced by the variable name) must be available in the root directory of the project. This should have been created with `setVariables`.

To normalise data tables, this function proceeds as follows:

1. Read in `input` and extract initial metadata from the file name.
2. Employ the function `tabshiftr::reorganise` to reshape `input` according to the respective schema description (see `tabshiftr::makeSchema`).
3. Match the territorial units in `input` via the `matchUnits`.
4. If ... has been provided with variables to match, those are matched via `matchVars`.
5. Harmonise territorial unit names.
6. If `update = TRUE`, store the processed data table at stage three.

Value

This function harmonises and integrates so far unprocessed data tables at stage two into stage three of the geospatial database. It produces for each nation in the registered data tables a comma-separated values file that includes all thematic areal data.

See Also

Other normalisers: `normGeometry()`

Examples

```
library(readr)

# build the example database
makeExampleDB(until = "normGeometry")

# normalise all available data tables, harmonising commodities
# according to the FAO commodity list ...
normTable(faoID = list(commodities = "target"), update = TRUE)

# ... and check the result
output <- read_csv(paste0(tempdir(), "/newDB/adb_tables/stage3/estonia.csv"))
```

`regDataseries` *Register a new dataseries*

Description

This function registers a new dataseries of both, geometries or areal data into the geospatial database.

Usage

```
regDataseries(
  name = NULL,
  description = NULL,
  homepage = NULL,
  licence_link = NULL,
  licence_path = NULL,
  notes = NULL,
  update = FALSE
)
```

Arguments

`name`	[`character(1)`] the dataseries abbreviation.
`description`	[`character(1)`] the "long name" or "brief description" of the dataseries.
`homepage`	[`character(1)`] the homepage of the data provider where the dataseries or additional information can be found.
`licence_link`	[`character(1)`] link to the licence or the webpage from which the licence was copied.
`licence_path`	[`character(1)`] path to the local file in with the licence text is stored.
`notes`	[`character(1)`] optional notes.
`update`	[`logical(1)`] whether or not the file 'inv_dataseries.csv' should be updated.

Value

Returns a tibble of the new entry that is appended to 'inv_dataseries.csv' in case `update = TRUE`.

Examples

```
# start the example database
makeExampleDB(until = "setVariables")

regDataseries(name = "gadm",
              description = "Database of Global Administrative Areas",
              homepage = "https://gadm.org/index.html",
              licence_link = "https://gadm.org/license.html",
              update = TRUE)
```

`regGeometry` *Register a new geometry entry*

Description

This function registers a new geometry of territorial units into the geospatial database.

Usage

```
regGeometry(
  nation = NULL,
  subset = NULL,
  gSeries = NULL,
  level = NULL,
  layer = NULL,
  nameCol = NULL,
  archive = NULL,
  archiveLink = NULL,
  nextUpdate = NULL,
  updateFrequency = NULL,
  notes = NULL,
  update = FALSE,
  overwrite = FALSE
)
```

Arguments

`nation`	[character(1)] either the nation name or the column of the file's attribute table that contains nations.
`subset`	[character(1)] optional argument to specify which subset the file contains. This could be a subset of territorial units (e.g. only one municipality) or of a target variable.
`gSeries`	[character(1)] the name of the geometry dataseries (see `regDataseries`).
`level`	[integerish(1)] the administrative level at which the geometry is recorded.

`layer` [character]
the name of the file's layer from which the geometry should be created (if applicable).

`nameCol` [character(.)]
the columns in which the names of administrative units are to be found, delimited by "|"; see Examples.

`archive` [character(1)]
the original file (perhaps a *.zip) from which the geometry emerges.

`archiveLink` [character(1)]
download-link of the archive.

`nextUpdate` [character(1)]
value describing the next anticipated update of this dataset (in YYYY-MM-DD format).

`updateFrequency`
[character(1)]
value describing the frequency with which the dataset is updated, according to the ISO 19115 Codelist, MD_MaintenanceFrequencyCode. Possible values are: 'continual', 'daily', 'weekly', 'fortnightly', 'quarterly', 'biannually', 'annually', 'asNeeded', 'irregular', 'notPlanned', 'unknown', 'periodic', 'semimonthly', 'biennially'.

`notes` [character(1)]
optional notes that are assigned to all features of this geometry.

`update` [logical(1)]
whether or not the file 'inv_geometries.csv' should be updated.

`overwrite` [logical(1)]
whether or not the geometry to register shall overwrite a potentially already existing older version.

Details

When processing geometries to which areal data shall be linked, carry out the following steps:

1. Determine the `nation`, a `subset` (if applicable), the dataseries of the geometry and the administrative `level`, and provide them as arguments to this function.
2. Run the function.
3. Export the shapefile with the following properties:
   - Format: GeoPackage
   - File name: What is provided as message by this function
   - CRS: EPSG:4326 - WGS 84
   - make sure that 'all fields are exported'
4. Confirm that you have saved the file.

Value

Returns a tibble of the entry that is appended to 'inv_geometries.csv' in case `update = TRUE`.

Examples

```
# build the example database
makeExampleDB(until = "regDataseries")

# The GADM dataset comes as *.zip archive
regGeometry(nation = "NAME_0",
            gSeries = "gadm",
            level = 1,
            layer = "example_geom1",
            nameCol = "NAME_0",
            archive = "example_geom.7z|example_geom1.gpkg",
            archiveLink = "https://gadm.org/",
            nextUpdate = "2019-10-01",
            updateFrequency = "quarterly",
            update = TRUE)

# The second administrative level in GADM contains names in the columns
# NAME_0 and NAME_1
regGeometry(nation = "NAME_0",
            gSeries = "gadm",
            level = 2,
            layer = "example_geom2",
            nameCol = "NAME_0|NAME_1",
            archive = "example_geom.7z|example_geom2.gpkg",
            archiveLink = "https://gadm.org/",
            nextUpdate = "2019-10-01",
            updateFrequency = "quarterly",
            update = TRUE)
```

---

regTable *Register a new areal data table*

Description

This function registers a new areal data table into the geospatial database.

Usage

```
regTable(
  nation = NULL,
  subset = NULL,
  dSeries = NULL,
  gSeries = NULL,
  level = NULL,
  begin = NULL,
  end = NULL,
  schema = NULL,
  archive = NULL,
  archiveLink = NULL,
```

```
  nextUpdate = NULL,
  updateFrequency = NULL,
  metadataLink = NULL,
  metadataPath = NULL,
  notes = NULL,
  update = FALSE,
  overwrite = FALSE
)
```

Arguments

nation	[character(1)] the nation for which the areal data are valid.
subset	[character(1)] optional argument to specify which subset the file contains. This could be a subset of territorial units (e.g. only one municipality) or of a target variable.
dSeries	[character(1)] the dataseries of the areal data (see regDataseries).
gSeries	[character(1)] optionally, the dataseries of the geometries, if the geometry dataseries deviates from the dataseries of the areal data (see regDataseries).
level	[integerish(1)] the administrative level at which the boundaries are recorded.
begin	[integerish(1)] the date from which on the data are valid.
end	[integerish(1)] the date until which the data are valid.
schema	[list(1)] the schema description of the table to read in (must have been placed in the global environment before calling it here.
archive	[character(1)] the original file from which the boundaries emerge.
archiveLink	[character(1)] download-link of the archive.
nextUpdate	[character(1)] when does the geometry dataset get updated the next time (format restricted to: YYYY-MM-DD).
updateFrequency	[character(1)] value describing the frequency with which the dataset is updated, according to the ISO 19115 Codelist, MD_MaintenanceFrequencyCode. Possible values are: 'continual', 'daily', 'weekly', 'fortnightly', 'quarterly', 'biannually', 'annually', 'asNeeded', 'irregular', 'notPlanned', 'unknown', 'periodic', 'semimonthly', 'biennially'.
metadataLink	[character(1)] if there is already metadata existing: link to the meta dataset.

metadataPath	[character(1)] if an existing meta dataset was downloaded along the data: the path where it is stored locally.
notes	[character(1)] optional notes.
update	[logical(1)] whether or not the file 'inv_tables.csv' should be updated.
overwrite	[logical(1)] whether or not the geometry to register shall overwrite a potentially already existing older version.

Details

When processing areal data tables, carry out the following steps:

1. Determine the nation, administrative level, a subset (if applicable) and the dataseries of the areal data and of the geometry, and provide them as arguments to this function.
2. Provide a begin and end date for the areal data.
3. Run the function.
4. (Re)Save the table with the following properties:
   - Format: csv
   - Encoding: UTF-8
   - File name: What is provided as message by this function
   - make sure that the file is not modified or reshaped. This will happen during data normalisation via the schema description, which expects the original table.
5. Confirm that you have saved the file.

Every areal data dataseries (dSeries) may come as a slight permutation of a particular table arrangement. The function normTable expects internally a schema description (a list that describes the position of the data components) for each data table, which is saved as paste0("meta_",dSeries,TAB_NUMBER). A template thereof, and documentation on how to set them up, can be found in tabshiftr::makeSchema with arealDB.

Value

Returns a tibble of the entry that is appended to 'inv_tables.csv' in case update = TRUE.

Examples

```
# build the example database
makeExampleDB(until = "regGeometry")

# the schema description for this table
library(tabshiftr)
schema_madeUp <- makeSchema(
  list(header = list(row = 1),
       variables = list(
         al1 =
```

```
            list(type = "id", col = 1),
          year =
            list(type = "id", col = 2),
          commodities =
            list(type = "id", col = 3),
          harvested =
            list(type = "measured", unit = "ha",
                 factor = 1, col = 4),
          production =
            list(type = "measured", unit = "t",
                 factor = 1, col = 5))))

regTable(nation = "estonia",
         subset = "soyMaize",
         dSeries = "madeUp",
         gSeries = "gadm",
         level = 1,
         begin = 1990,
         end = 2017,
         schema = schema_madeUp,
         archive = "example_table.7z|example_table1.csv",
         archiveLink = "...",
         nextUpdate = "2019-10-01",
         updateFrequency = "quarterly",
         metadataLink = "...",
         metadataPath = "my/local/path",
         update = TRUE)
```

`setPath` *Set the root path*

Description

Initiate a geospatial database or register a database that exists at the root path.

Usage

```
setPath(root = NULL)
```

Arguments

`root`	`[character(1)]` path to the root directory that contains or shall contain an areal database.

Details

This is the first function that is run in a project, as it initiates the areal database by creating the default sub-directories and initial inventory tables. When a database has already been set up, this function is used to register that path in the options of the current R session.

Value

No return value, called for the side effect of creating the directory structure of the new areal database and a new environment that contains the database metadata.

Examples

```
setPath(root = paste0(tempdir(), "/newDB"))
```

`setVariables` *Set index and translation tables*

Description

Use a pre-compiled table to create an index and/or translation table for the target variables of an areal database.

Usage

```
setVariables(
  input = NULL,
  variable = NULL,
  type = "both",
  pid = NULL,
  origin = NULL,
  target = NULL
)
```

Arguments

`input`	[`tibble(1)`] a possibly already existing table based on which the output should be constructed; see details.
`variable`	[`character(1)`] name of the variable and thus of the output file(s).
`type`	[`character(1)`] the type of table to create, either an index (`"id"`), a translation table (`"tt"`) or `"both"` (default).
`pid`	[`character(1)`] column in `input` that contains the primary ID for the index table. If this is not given, an ID name is created as `paste0(str_sub(variable,1,3),"ID")`
`origin`	[`character(1)`] column in `input` that contains terms that shall be translated.
`target`	[`character(1)`] column in `input` that contains the standardised terms.

Details

This is the second function that is run in a project, as it creates index and translation tables for the target variables that shall be stored in an areal database.

- An index table relates an ID (given in `pid`) to the variable terms (given in `target`) and potentially to ancillary information. Such tables should be compiled before a project is started and should contain a clear set of values per variable (which will be used as standard ontology).
- A translation table relates terms in foreign languages (given in `origin`) to terms in the target language (given in `target`). If target does not exist, the terms are simply registered as "original" to be used for fuzzy matching.

Value

No return value, called for the side effect of writing a table to the project root directory with name `paste0(type,"_",variable,".csv")`.

Examples

```
library(readr)
inPath <- system.file("test_datasets",
                      package = "arealDB",
                      mustWork = TRUE)

# start the example database
makeExampleDB(until = "setPath")

# create index from an already existing table
comm <- read_csv(file = paste0(inPath, "/id_commodities.csv"),
                 col_types = "iccc")
setVariables(input = comm, variable = "commodities",
             pid = "faoID", target = "simpleName")
```

---

testCompressed *Test whether a file is a compressed file*

Description

Test whether a file is a compressed file

Usage

```
testCompressed(x)
```

Arguments

`x`	[`character(1)`] the file name.

Details

This function looks at the file-extension and if it is one of `.gz`, `.bz2`, `.tar .zip`, `.tgz`, `.gzip` or `.7z`, it returns the value `TRUE`.

`translateTerms` *Translate terms*

Description

Translate terms based on fuzzy matching.

Usage

```
translateTerms(
  terms,
  index = NULL,
  source = NULL,
  strict = FALSE,
  fuzzy_terms = NULL,
  fuzzy_dist = 5,
  inline = TRUE,
  verbose = TRUE
)
```

Arguments

`terms`	[`character(.)`] terms to be translated.
`index`	[`character(1)`] name of a table that contains translations.
`source`	[`named list(1)`] the table or geometry ID from which the terms have been taken. List must be named with either `tabID` or `geoID` to denote where the ID comes from.
`strict`	[`logical(1)`] whether or not to stick to the terms that have been defined as `'original'` in a translation table.
`fuzzy_terms`	[`vector(.)`] additional target terms with which a fuzzy match should be carried out.
`fuzzy_dist`	[`integerish(1)`] the maximum edit-distance for which terms of fuzzy-matching should be suggested as match.
`inline`	[`logical(1)`] whether or not to edit translations inline in R (only possible in linux), or in the 'translating.csv' in your database root directory.
`verbose`	[`logical(1)`] be verbose about what is happening (default `TRUE`).

Details

This is basically a sophisticated matching algorithm, that adds new entries to the respective index.

Value

A table of translated `terms`.

---

updateTable *Update a table*

Description

Update any inventory, index or translation table of a geospatial database.

Usage

```
updateTable(index = NULL, name = NULL)
```

Arguments

`index`	[`tibble(1)`] the table to use as update.
`name`	[`character(1)`] name of the table that shall be updated.

View_sf *View large simple features*

Description

View large simple features

Usage

```
View_sf(x)
```

Arguments

`x`	[`sf`] the sf object to view.

Index

∗ **datasets**
countries, 2
∗ **normalisers**
normGeometry, 5
normTable, 8

countries, 2, *6*

getColTypes, 3

makeExampleDB, 3
makeSchema, *9*, *15*
matchUnits, 4, *9*
matchVars, 5, *9*

normGeometry, 5, *9*
normTable, *7*, 8, *15*

regDataseries, 10, *11*, *14*
regGeometry, 11
regTable, 13
reorganise, *9*

setPath, 16
setVariables, *5*, *9*, 17
st_drivers, *6*
suppressMessages, *6*, *8*

testCompressed, 18
translateTerms, 19

updateTable, 20

View_sf, 20