Package ‘MazamaLocationUtils’

November 20, 2019

Type Package

Version 0.1.6

Title Manage Spatial Metadata for Known Locations

Author Jonathan Callahan [aut, cre]

Maintainer Jonathan Callahan <jonathan.s.callahan@gmail.com>

Description A suite of utility functions for discovering and managing metadata associated with sets of spatially unique ``known locations''.

License GPL-3

URL https://github.com/MazamaScience/MazamaCoreUtils

BugReports https://github.com/MazamaScience/MazamaCoreUtils/issues

Depends R (>= 3.1.0)

Imports digest, dplyr, geodist, httr, lubridate, methods, magrittr, MazamaCoreUtils, MazamaSpatialUtils, readr, revgeo, rlang, stringr

Suggests knitr, markdown, testthat (>= 2.1.0), rmarkdown, roxygen2

Encoding UTF-8

VignetteBuilder knitr

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Repository CRAN

Date/Publication 2019-11-20 16:40:02 UTC

R topics documented:

`coreMetadataNames` *Names of standard spatial metadata columns*

Description

Character string identifiers of the different types of spatial metadata this package can generate.

Usage

```
coreMetadataNames
```

Format

A vector with 3 elements

Details

coreMetadataNames

`getLocationDataDir` *Get location data directory*

Description

Returns the directory path where known location data tables are located.

Usage

```
getLocationDataDir()
```

Value

Absolute path string.

See Also

LocationDataDir

setLocationDataDir

`id_monitors_500` *Idaho monitor locations dataset*

Description

The `id_monitor_500` dataset provides a set of known locations associated with Idaho state air quality monitors. This dataset was generatedon 2019-10-21 by running:

```
library(PWFSLSmoke)
library(MazamaLocationUtils)

mazama_initialize()
setLocationDataDir("./data")

monitor <- monitor_loadLatest()
lons <- monitor$meta$longitude
lats <- monitor$meta$latitude

table_initialize()
  table_addLocation(lons, lats, radius = 500)
  table_save("id_monitors_500")
```

Usage

```
id_monitors_500
```

Format

A tibble with 34 rows and 13 columns of data.

See Also

or_monitors_500

wa_monitors_500

LocationDataDir *Directory for location data*

Description

This package maintains an internal directory path which users can set using `setLocationDataDir()`. All package functions use this directory whenever known location tables are accessed.

The default setting when the package is loaded is `getwd()`.

Format

Absolute path string.

See Also

getLocationDataDir

setLocationDataDir

---

location_createID *Create one or more unique locationIDs*

Description

A unique locationID is created for each incoming `longitude` and `latitude`. The following code is used to generate each locationID. See the references for details.

```
# Retain accuracy up to ~.1m
locationString <- paste0(
  sprintf("
  "_",
  sprintf("
)

# Avoid collisions until billions of records
locationID <- digest::digest(locationString, algo = "xxhash64")
```

Usage

```
location_createID(longitude = NULL, latitude = NULL)
```

Arguments

`longitude`	Single longitude in decimal degrees E, Default: NULL
`latitude`	Single latitude in decimal degrees N, Default: NULL

Value

Vector of character locationIDs.

References

https://en.wikipedia.org/wiki/Decimal_degrees

https://www.johndcook.com/blog/2017/01/10/probability-of-secure-hash-collisions/

Examples

```
# Wenatchee
lon <- -120.325278
lat <- 47.423333
locationID <- location_createID(lon, lat)
```

`location_getSingleAddress_Photon`

Get address data from the Photon API to OpenStreetMap

Description

The Photon API is used get address data associated with the `longitude` and `latitude`. The following list of data is returned:

- `houseNumber`
- `street`
- `city`
- `stateCode`
- `stateName`
- `zip`
- `countryCode`
- `countryName`

The function makes an effort to convert both `state` and `country` `Name` into `Code` with codes defaulting to `NA`. Both `Name` and `Code` are returned so that improvements can be made in the conversion algorithm.

Usage

```
location_getSingleAddress_Photon(longitude = NULL, latitude = NULL,
  verbose = TRUE)
```

Arguments

longitude	Single longitude in decimal degrees E, Default: NULL
latitude	Single latitude in decimal degrees N, Default: NULL
verbose	Logical controlling the generation of progress messages.

Value

List of address components.

References

http://photon.komoot.de

Examples

```
# Set up standard directories and spatial data
spatialDataDir <- tempdir() # typically "~/Data/Spatial"
mazama_initialize(spatialDataDir)

# Wenatchee
lon <- -120.325278
lat <- 47.423333
addressList <- location_getSingleAddress_Photon(lon, lat)
str(addressList)
```

---

location_getSingleElevation_USGS

Get elevation data from a USGS web service

Description

USGS APIs are used to determine the elevation associated with the `longitude` and `latitude`.

Usage

```
location_getSingleElevation_USGS(longitude = NULL, latitude = NULL,
  verbose = TRUE)
```

Arguments

`longitude`	Single longitude in decimal degrees E, Default: NULL
`latitude`	Single latitude in decimal degrees N, Default: NULL
`verbose`	Logical controlling the generation of progress messages.

Value

Numeric elevation value.

References

https://nationalmap.gov/epqs/

Examples

```
# Wenatchee
lon <- -120.325278
lat <- 47.423333
location_getSingleElevation_USGS(lon, lat)
```

`location_initialize` *Create known location record with core metadata*

Description

Creates a known location record with the following columns of core metadata:

- locationID
- locationName
- longitude
- latitude
- elevation
- countryCode
- stateCode
- county
- timezone
- houseNumber
- street
- city
- zip

Usage

```
location_initialize(longitude = NULL, latitude = NULL,
  stateDataset = "NaturalEarthAdm1", verbose = TRUE)
```

Arguments

longitude	Single longitude in decimal degrees E, Default: NULL
latitude	Single latitude in decimal degrees N, Default: NULL
stateDataset	Name of spatial dataset to use for determining state
verbose	Logical controlling the generation of progress messages.

Value

Tibble with a single new known location.

Examples

```
# Set up standard directories and spatial data
spatialDataDir <- tempdir() # typically "~/Data/Spatial"
mazama_initialize(spatialDataDir)

# Wenatchee
lon <- -120.325278
lat <- 47.423333
locationRecord <- location_initialize(lon, lat)
```

MazamaLocationUtils *Manage Spatial Metadata for Known Locations*

Description

A suite of utility functions for discovering and managing metadata associated with sets of spatially unique "known locations".

This package is intended to be used in support of data management activities associated with fixed locations in space. The motivating fields include both air and water quality monitoring where fixed sensors report at regular time intervals.

Details

When working with environmental monitoring time series, one of the first things you have to do is create unique identifiers for each individual time series. In an ideal world, each environmental time series would have both a `locationID` and a `sensorID` that uniquely identify the spatial location and specific instrument making measurements. A unique `timeseriesID` could be produced as `locationID_sensorID`. Metadata associated with each time series would contain basic information needed for downstream analysis including at least:

```
timeseriesID, locationID, sensorID, longitude, latitude, ...
```

- Multiple sensors placed at a location could be be grouped by `locationID`.
- An extended timeservers for a mobile sensor would group by `sensorID`.
- Maps would be created using `longitude,latitude`.
- Time series would be accessed from a secondary `data` table with `timeseriesID`.

Unfortunately, we are rarely supplied with a truly unique and truly spatial `locationID`. Instead we often use `sensorID` or an associated non-spatial identifier as a standin for `locationID`.

Complications we have seen include:

- GPS-reported longitude and latitude can have _jitter_ in the fourth or fifth decimal place making it challenging to use them to create a unique `locationID`.
- Sensors are sometimes _repositioned_ in what the scientist considers the "same location".
- Data for a single sensor goes through different processing pipelines using different identifiers and is later brought together as two separate timeseries.
- The radius of what constitutes a "single location" depends on the instrumentation and scientific question being asked.
- Deriving location-based metadata from spatial datasets is computationally intensive unless saved and identified with a unique `locationID`.
- Automated searches for spatial metadata occasionally produce incorrect results because of the non-infinite resolution of spatial datasets.

This package attempts to address all of these issues by maintaining a table of known locations for which CPU intensive spatial data calculations have already been performed. While requests to add new locations to the table may take some time, searches for spatial metadata associated with existing locations are simple lookups.

Working in this manner will solve the problems initially mentioned but also provides further useful functionality.

- Administrators can correct entries in the `collectionName` table. (_e.g._ locations in river bends that even high resolution spatial datasets mis-assign)
- Additional, non-automatable metadata can be added to `collectionName`. (_e.g._ commonly used location names within a community of practice)
- Different field campaigns can have separate `collectionName` tables.
- `.csv` or `.rda` versions of well populated tables can be downloaded from a URL and used locally, giving scientists working with known locations instant access to spatial data that otherwise requires special skills, large datasets and lots of compute cycles.

mazama_initialize *Initialize with MazamaScience standard directories*

Description

Convenience function to initialize spatial data. Wraps the following setup lines:

```
MazamaSpatialUtils::setSpatialDataDir(spatialDataDir)

MazamaSpatialUtils::loadSpatialData("EEZCountries")
MazamaSpatialUtils::loadSpatialData("OSMTimezones")
MazamaSpatialUtils::loadSpatialData("NaturalEarthAdm1")
MazamaSpatialUtils::loadSpatialData("USCensusCounties")
```

Usage

```
mazama_initialize(spatialDataDir = "~/Data/Spatial")
```

Arguments

spatialDataDir	Directory where spatial datasets are found, Default: "~/Data/Spatial"

Value

No return value.

Examples

```
# Set up directory for spatial data
spatialDataDir <- tempdir() # typically "~/Data/Spatial"
MazamaSpatialUtils::setSpatialDataDir(spatialDataDir)

# Install core spatial datasets (168 MB download)
MazamaSpatialUtils::installSpatialData()

exists("NaturalEarthAdm1")
mazama_initialize(spatialDataDir)
exists("NaturalEarthAdm1")
class(NaturalEarthAdm1)
```

or_monitors_500 *Oregon monitor locations dataset*

Description

The or_monitor_500 dataset provides a set of known locations associated with Oregon state air quality monitors. This dataset was generatedon 2019-10-21 by running:

```
library(PWFSLSmoke)
library(MazamaLocationUtils)

mazama_initialize()
setLocationDataDir("./data")

monitor <- monitor_loadLatest()
lons <- monitor$meta$longitude
lats <- monitor$meta$latitude

table_initialize()
  table_addLocation(lons, lats, radius = 500)
  table_save("or_monitors_500")
```

Usage

```
or_monitors_500
```

Format

A tibble with 40 rows and 13 columns of data.

See Also

id_monitors_500

wa_monitors_500

setLocationDataDir *Set location data directory*

Description

Sets the data directory where known location data tables are located. If the directory does not exist, it will be created.

Usage

```
setLocationDataDir(dataDir)
```

Arguments

dataDir	Directory where location tables are stored.

Value

Silently returns previous value of the data directory.

See Also

LocationDataDir

getLocationDataDir

---

table_addColumn — *Add a new column of metadata to a table*

Description

A new metadata column is added to the `locationTbl`. For matching `locationID` records the assaociated `locatioData` is inserted. Otherwise, the new column will be initialized with `NA`.

Usage

```
table_addColumn(locationTbl = NULL, columnName = NULL,
  locationID = NULL, locationData = NULL, verbose = TRUE)
```

Arguments

locationTbl	Tibble of known locations, Default: NULL
columnName	Name to use for the new column, Default: NULL
locationID	Vector of `locationID` strings, Default: NULL
locationData	Vector of data to used at matching records, Default: NULL
verbose	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_removeColumn

table_updateColumn

Examples

```
# Starting table
locationTbl <- get(data("wa_monitors_500"))
names(locationTbl)

# Add an empty column
locationTbl <-
  locationTbl %>%
  table_addColumn("siteName")

names(locationTbl)
```

---

table_addLocation *Add new known location records to a table*

Description

Incoming `longitude` and `latitude` values are compared against the incoming `locationTbl` to see if the are already within `radius` meters of an existing entry. A new record is created for each location that is not already found in `locationTbl`.

Usage

```
table_addLocation(locationTbl = NULL, longitude = NULL,
  latitude = NULL, radius = NULL, stateDataset = "NaturalEarthAdm1",
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`longitude`	Vector of longitudes in decimal degrees E, Default: NULL
`latitude`	Vector of latitudes in decimal degrees N, Default: NULL
`radius`	Radius in meters, Default: NULL
`stateDataset`	Name of spatial dataset to use for determining state codes, Default: 'NaturalEarthAdm1'
`verbose`	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

Note

This funciton is a vecorized version of `table_addSingleLocation()`.

See Also

table_addSingleLocation

table_removeRecord

table_updateSingleRecord

Examples

```
# Set up standard directories and spatial data
spatialDataDir <- tempdir() # typically "~/Data/Spatial"
mazama_initialize(spatialDataDir)

locationTbl <- get(data("wa_monitors_500"))

# Coulee City, WA
lon <- -119.290904
lat <- 47.611942

locationTbl <-
  locationTbl %>%
  table_addLocation(lon, lat, radius = 500)
```

`table_addSingleLocation` *Add a single new known location record to a table*

Description

Incoming `longitude` and `latitude` values are compared against the incoming `locationTbl` to see if the are already within `radius` meters of an existing entry. A new record is created for if the location is not already found in `locationTbl`.

Usage

```
table_addSingleLocation(locationTbl = NULL, longitude = NULL,
  latitude = NULL, radius = NULL, stateDataset = "NaturalEarthAdm1",
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`longitude`	Single longitude in decimal degrees E, Default: NULL
`latitude`	Single latitude in decimal degrees N, Default: NULL
`radius`	Radius in meters, Default: NULL
`stateDataset`	Name of spatial dataset to use for determining state codes, Default: 'NaturalEarthAdm1'
`verbose`	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_addLocation

table_removeRecord

table_updateSingleRecord

Examples

```
# Set up standard directories and spatial data
spatialDataDir <- tempdir() # typically "~/Data/Spatial"
MazamaSpatialUtils::setSpatialDataDir(spatialDataDir)

locationTbl <- get(data("wa_monitors_500"))

# Coulee City, WA
lon <- -119.290904
lat <- 47.611942

locationTbl <-
  locationTbl %>%
  table_addSingleLocation(lon, lat, radius = 500)
```

`table_export` *Export a known location table*

Description

Export a known location tibble as CSV format.

Usage

```
table_export(locationTbl = NULL, outputType = "csv")
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`outputType`	Output format, Default: 'csv'

Value

Representation of a known location table in the desired format.

Examples

```
locationTbl <- get(data("wa_monitors_500"))
csvString <- table_export(locationTbl)
```

---

table_getLocationID *Return IDs of known locations*

Description

Returns a vector of locationIDs for the known locations that each incoming location will be assigned to within the given. If more than one known location exists within the given radius, the closest will be assigned. NA will be returned for each incoming that cannot be assigend to a known location in locationTbl.

Usage

```
table_getLocationID(locationTbl = NULL, longitude = NULL,
  latitude = NULL, radius = NULL)
```

Arguments

locationTbl	Tibble of known locations, Default: NULL
longitude	Vector of longitudes in decimal degrees E, Default: NULL
latitude	Vector of latitudes in decimal degrees N, Default: NULL
radius	Radius in meters, Default: NULL

Value

Vector of known locationIDs.

Examples

```
locationTbl <- get(data("wa_monitors_500"))

# Wenatchee
lon <- -120.325278
lat <- 47.423333

# Too small a radius will not find a match
table_getLocationID(locationTbl, lon, lat, radius = 50)

# Expanding the radius will find one
table_getLocationID(locationTbl, lon, lat, radius = 5000)
```

`table_getNearestDistance` *Return distances to nearest known locations*

Description

Returns a distances from known locations in `locationTbl`, one for each incoming location. If no known location is found within `radius` meters for a particular incoming location, that distance in the vector will be `NA`.

Usage

```
table_getNearestDistance(locationTbl = NULL, longitude = NULL,
  latitude = NULL, radius = NULL)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`longitude`	Vector of longitudes in decimal degrees E, Default: NULL
`latitude`	Vector of latitudes in decimal degrees N, Default: NULL
`radius`	Radius in meters, Default: NULL

Value

Vector of distances from known locations.

`table_getNearestLocation` *Return known locations*

Description

Returns a tibble of known locations from `locationTbl`, one for each incoming location. If no known location is found for a particular incoming location, that record in the tibble will contain all `NA`.

Usage

```
table_getNearestLocation(locationTbl = NULL, longitude = NULL,
  latitude = NULL, radius = NULL)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`longitude`	Vector of longitudes in decimal degrees E, Default: NULL
`latitude`	Vector of latitudes in decimal degrees N, Default: NULL
`radius`	Radius in meters, Default: NULL

Value

Tibble of known locations.

Examples

```
locationTbl <- get(data("wa_monitors_500"))

# Wenatchee
lon <- -120.325278
lat <- 47.423333

# Too small a radius will not find a match
table_getNearestLocation(locationTbl, lon, lat, radius = 50) %>% str()

# Expanding the radius will find one
table_getNearestLocation(locationTbl, lon, lat, radius = 5000) %>% str()
```

`table_getRecordIndex` *Return indexes of known location records*

Description

Returns a vector of `locationTbl` row indexes for the locations associated with each `locationID`.

Usage

```
table_getRecordIndex(locationTbl = NULL, locationID = NULL,
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`locationID`	Vector of `locationID` strings, Default: NULL
`verbose`	Logical controlling the generation of progress messages.

Value

Vector of `locationTbl` row indexes.

Examples

```
locationTbl <- get(data("wa_monitors_500"))

# Wenatchee
lon <- -120.325278
lat <- 47.423333

# Get the locationID first
locationID <- table_getLocationID(locationTbl, lon, lat, radius = 5000)

# Now find the row associated with this ID
recordIndex <- table_getRecordIndex(locationTbl, locationID)

str(locationTbl[recordIndex,])
```

---

table_initialize *Create an empty known location table*

Description

Creates an empty known location tibble with the following columns of core metadata:

- locationID
- locationName
- longitude
- latitude
- elevation
- countryCode
- stateCode
- county
- timezone
- houseNumber
- street
- city
- zip

Usage

```
table_initialize()
```

Value

Empty known location tibble with the specified metadata columns.

Examples

```
# Create an empty Tbl
emptyTbl <- table_initialize()
print(emptyTbl)
```

`table_load` *Load a known location table*

Description

Load a tibble of known locations from the preferred directory.

Usage

```
table_load(collectionName = NULL)
```

Arguments

`collectionName`	Character identifier for this table, Default: NULL

Value

Tibble of known locations.

See Also

setLocationDataDir

Examples

```
# Set the directory for saving location tables
setLocationDataDir(tempdir())

# Load an example table and check the dimensions
locationTbl <- get(data("wa_monitors_500"))
dim(locationTbl)

# Save it as "table_load_example"
table_save(locationTbl, "table_load_example")

# Load it and check the dimensions
my_table <- table_load("table_load_example")
dim(my_table)

# Check the locationDataDir
list.files(getLocationDataDir(), pattern = "table_load_example")
```

table_removeColumn — *Remove a column of metadata in a table*

Description

Remove the column matching `columnName`. This function can be used in pipelines.

Usage

```
table_removeColumn(locationTbl = NULL, columnName = NULL,
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`columnName`	Name of the colun to be removed, Default: NULL
`verbose`	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_addColumn

table_removeColumn

Examples

```
# Starting table
locationTbl <- get(data("wa_monitors_500"))
names(locationTbl)

# Add a new column
locationTbl <-
  locationTbl %>%
  table_addColumn("siteName")

names(locationTbl)

# Now remove it
locationTbl <-
  locationTbl %>%
  table_removeColumn("siteName")

names(locationTbl)

## Not run:
```

```
# Cannot remove "core" metadata
locationTbl <-
  locationTbl %>%
  table_removeColumn("zip")

## End(Not run)
```

`table_removeRecord` *Remove location records from a table*

Description

Incoming `locationID` values are compared against the incoming `locationTbl` and any matches are removed.

Usage

```
table_removeRecord(locationTbl = NULL, locationID = NULL,
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`locationID`	Vector of `locationID` strings, Default: NULL
`verbose`	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_addLocation

table_addSingleLocation

table_updateSingleRecord

Examples

```
locationTbl <- get(data("wa_monitors_500"))
dim(locationTbl)

# Wenatchee
lon <- -120.325278
lat <- 47.423333

# Get the locationID first
locationID <- table_getLocationID(locationTbl, lon, lat, radius = 500)
```

```
# Remove it
locationTbl <- table_removeRecord(locationTbl, locationID)
dim(locationTbl)

# Test
table_getLocationID(locationTbl, lon, lat, radius = 500)
```

---

table_save *Save a known location table*

Description

Save a tibble of known locations to the preferred directory.

Usage

```
table_save(locationTbl = NULL, collectionName = NULL, backup = TRUE,
  outputType = "rda")
```

Arguments

locationTbl	Tibble of known locations, Default: NULL
collectionName	Character identifier for this table, Default: NULL
backup	Logical specifying whether to save a backup version of any existing tables sharing collectionName.
outputType	Output format, Default: 'rda'

Details

Backup files are saved with "YYYY-mm-ddTHH:MM:SS"

Value

File path of saved file.

Examples

```
# Set the directory for saving location tables
setLocationDataDir(tempdir())

# Load an example table and check the dimensions
locationTbl <- get(data("wa_monitors_500"))
dim(locationTbl)

# Save it as "table_save_example"
table_save(locationTbl, "table_save_example")

# Add a column and save again
```

```
locationTbl %>%
  table_addColumn("my_column") %>%
  table_save(locationTbl, "table_save_example")

# Check the locationDataDir
list.files(getLocationDataDir(), pattern = "table_save_example")
```

---

table_updateColumn *Update a column of metadata in a table*

Description

For matching locationID records the assaociated locatioData is used to replace any existing value in columnName.

Usage

```
table_updateColumn(locationTbl = NULL, columnName = NULL,
  locationID = NULL, locationData = NULL, verbose = TRUE)
```

Arguments

locationTbl	Tibble of known locations, Default: NULL
columnName	Name to use for the new column, Default: NULL
locationID	Vector of locationID strings, Default: NULL
locationData	Vector of data to used at matching records, Default: NULL
verbose	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_addColumn

table_removeColumn

Examples

```
locationTbl <- get(data("wa_monitors_500"))
wa <- get(data("wa_airfire_meta"))

# We will merge some metadata from wa into locationTbl

# Record indices for wa
wa_indices <- seq(5,65,5)
wa_sub <- wa[wa_indices,]
```

```
locationID <- table_getLocationID(locationTbl, wa_sub$longitude, wa_sub$latitude, radius = 500)
locationData <- wa_sub$siteName

locationTbl <- table_updateColumn(locationTbl, "siteName", locationID, locationData)

# Look at the data we attempted to merge
wa$siteName[wa_indices]

# And two columns from the updated locationTbl
locationTbl_indices <- table_getRecordIndex(locationTbl, locationID)
locationTbl[locationTbl_indices, c("city","siteName")]
```

---

table_updateSingleRecord   *Update a single known location record in a table*

Description

Information in the `locationList` is used to replace existing information found in `locationTbl`. This function can be used for small tweaks to an existing `locationTbl`. Wholesale replacement of records should be performed with `table_removeRecord()` followed by `table_addLocation()`.

Usage

```
table_updateSingleRecord(locationTbl = NULL, locationList = NULL,
  verbose = TRUE)
```

Arguments

`locationTbl`	Tibble of known locations, Default: NULL
`locationList`	List containing ‘locationID‘ and one or more named columns whose values are to be replaced, Default: NULL
`verbose`	Logical controlling the generation of progress messages.

Value

Updated tibble of known locations.

See Also

table_addLocation

table_addSingleLocation

table_removeRecord

Examples

```
locationTbl <- get(data("wa_monitors_500"))

# Wenatchee
wenatcheeRecord <-
  locationTbl %>%
  dplyr::filter(city == "Wenatchee")

str(wenatcheeRecord)

wenatcheeID <- wenatcheeRecord$locationID

locationTbl <- table_updateSingleRecord(
  locationTbl,
  locationList = list(
    locationID = wenatcheeID,
    locationName = "Wenatchee-Fifth St"
  )
)

# Look at the new record
locationTbl %>%
  dplyr::filter(city == "Wenatchee") %>%
  str()
```

`validateLonLat` *Validate longitude and latitude values*

Description

Longitude and latitude are validated to be parseable as numeric and within the bounds -180:180 and -90:90. If validation fails, an error is generated.

Usage

```
validateLonLat(longitude = NULL, latitude = NULL)
```

Arguments

`longitude`	Single longitude in decimal degrees E, Default: NULL
`latitude`	Single latitude in decimal degrees N, Default: NULL

Value

Invisibly returns `TRUE` if no error message has been generated.

`validateLonsLats` *Validate longitude and latitude vectors*

Description

Longitude and latitude vectors validated to be parseable as numeric and within the bounds -180:180 and -90:90. If validation fails, an error is generated.

Usage

```
validateLonsLats(longitude = NULL, latitude = NULL)
```

Arguments

`longitude`	Vector of longitudes in decimal degrees E, Default: NULL
`latitude`	Vector of latitudes in decimal degrees N, Default: NULL

Value

Invisibly returns `TRUE` if no error message has been generated.

`validateMazamaSpatialUtils` *Validate proper setup of MazamaSpatialUtils*

Description

The **MazamaSpatialUtils** package mus be properly installed and initialized before using functions from the **MazamaLocationUtils** package. Functions can test for this

Usage

```
validateMazamaSpatialUtils()
```

Value

Invisibly returns `TRUE` if no error message has been generated.

wa_airfire_meta *Washington monitor metadata dataset*

Description

The wa_pwfsl_meta dataset provides a set of Washington state air quality monitor metadata used by the USFS AirFire group. This dataset was generatedon 2019-10-21 by running:

```
library(PWFSLSmoke)

wa_airfire_meta <-
  monitor_loadLatest()
  monitor_subset(stateCodes = "WA")
  monitor_extractMeta()

save(wa_airfire_meta, file = "data/wa_airfire_meta.rda")
```

Usage

```
wa_airfire_meta
```

Format

A tibble with 69 rows and 19 columns of data.

wa_monitors_500 *Wshington monitor locations dataset*

Description

The wa_monitor_500 dataset provides a set of known locations associated with Washington state air quality monitors. This dataset was generatedon 2019-10-21 by running:

```
library(PWFSLSmoke)
library(MazamaLocationUtils)

mazama_initialize()
setLocationDataDir("./data")

monitor <- monitor_loadLatest()
lons <- monitor$meta$longitude
lats <- monitor$meta$latitude

table_initialize()
  table_addLocation(lons, lats, radius = 500)
  table_save("wa_monitors_500")
```

Usage

```
wa_monitors_500
```

Format

A tibble with 69 rows and 13 columns of data.

See Also

id_monitors_500

or_monitors_500

Index