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, htttr, 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 .............................................. 2
getLocationDataDir ............................................. 3
id_monitors_500 ................................................ 3
LocationDataDir .................................................. 4
coreMetadataNames

**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

---

Index 30

<table>
<thead>
<tr>
<th>Function</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>location_createID</td>
<td>4</td>
</tr>
<tr>
<td>location_getSingleAddress_Photon</td>
<td>5</td>
</tr>
<tr>
<td>location_getSingleElevation_USGS</td>
<td>6</td>
</tr>
<tr>
<td>location_initialize</td>
<td>7</td>
</tr>
<tr>
<td>MazamaLocationUtils</td>
<td>8</td>
</tr>
<tr>
<td>mazama_initialize</td>
<td>9</td>
</tr>
<tr>
<td>or_monitors_500</td>
<td>10</td>
</tr>
<tr>
<td>setLocationDataDir</td>
<td>11</td>
</tr>
<tr>
<td>table_addColumn</td>
<td>12</td>
</tr>
<tr>
<td>table_addLocation</td>
<td>13</td>
</tr>
<tr>
<td>table_addSingleLocation</td>
<td>14</td>
</tr>
<tr>
<td>table_export</td>
<td>15</td>
</tr>
<tr>
<td>table_getLocationID</td>
<td>16</td>
</tr>
<tr>
<td>table_getNearestDistance</td>
<td>17</td>
</tr>
<tr>
<td>table_getNearestLocation</td>
<td>17</td>
</tr>
<tr>
<td>table_getRecordIndex</td>
<td>18</td>
</tr>
<tr>
<td>table_initialize</td>
<td>19</td>
</tr>
<tr>
<td>table_load</td>
<td>20</td>
</tr>
<tr>
<td>table_removeColumn</td>
<td>21</td>
</tr>
<tr>
<td>table_removeRecord</td>
<td>22</td>
</tr>
<tr>
<td>table_save</td>
<td>23</td>
</tr>
<tr>
<td>table_updateColumn</td>
<td>24</td>
</tr>
<tr>
<td>table_updateSingleRecord</td>
<td>25</td>
</tr>
<tr>
<td>validateLonLat</td>
<td>26</td>
</tr>
<tr>
<td>validateLonsLats</td>
<td>27</td>
</tr>
<tr>
<td>validateMazamaSpatialUtils</td>
<td>27</td>
</tr>
<tr>
<td>wa_airfire_meta</td>
<td>28</td>
</tr>
<tr>
<td>wa_monitors_500</td>
<td>28</td>
</tr>
</tbody>
</table>
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 generated on 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.

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

# Avoid collisions until billions of records
d_locationID <- digest::digest(d_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_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

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_initialize**

Create known location record with core metadata

### Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>longitude</td>
<td>Single longitude in decimal degrees E, Default: NULL</td>
</tr>
<tr>
<td>latitude</td>
<td>Single latitude in decimal degrees N, Default: NULL</td>
</tr>
<tr>
<td>verbose</td>
<td>Logical controlling the generation of progress messages.</td>
</tr>
</tbody>
</table>

### Value

Numeric elevation value.

### References

https://nationalmap.gov/epqs/

### Examples

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

---

**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

```r
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

```r
# 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)
```

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 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)
The `or_monitors_500` dataset provides a set of known locations associated with Oregon state air quality monitors. This dataset was generated on 2019-10-21 by running:

```r
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)`
**table_addColumn**

**Arguments**
- `dataDir` Directory where location tables are stored.

**Value**
Silently returns previous value of the data directory.

**See Also**
- `LocationDataDir`
- `getLocationDataDir`

---

**Description**
A new metadata column is added to the `locationTbl`. For matching `locationID` records the associated locationData is inserted. Otherwise, the new column will be initialized with NA.

**Usage**

```r
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 they 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 function is a vectorized version of `table_addSingleLocation()`.
See Also

- `table_addSingleLocation`
- `table_removeRecord`
- `table_updateSingleRecord`

Examples

```r
# 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)
```

Description

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

Usage

```r
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.
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

```r
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 assigned to a known location in locationTbl.

Usage

```r
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

```r
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

```r
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

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

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()
```

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

```r
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,])
```

### 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

```r
table_initialize()
```

### Value

Empty known location tibble with the specified metadata columns.
Examples

```r
# 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

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

Arguments

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

Value

Tibble of known locations.

See Also

- `setLocationDataDir`

Examples

```r
# 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**
```r
table_removeColumn(locationTbl = NULL, columnName = NULL,  
                    verbose = TRUE)
```

**Arguments**
- `locationTbl` Tibble of known locations, Default: NULL
- `columnName` Name of the column 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**
```r
# 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
table_removeRecord <-
  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**

```r
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**

```r
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

```r
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

```r
# 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
```
table_updateColumn

**Description**
For matching `locationID` records the associated `locationData` is used to replace any existing value in `columnName`.

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

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

```r
# We will merge some metadata from wa into locationTbl

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

Examples

```r
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**

```r
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 must 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.
Description

The `wa_monitors_500` dataset provides a set of known locations associated with Washington state air quality monitors. This dataset was generated on 2019-10-21 by running:

```r
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 19 columns of data.
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

∗Topic datasets
  coreMetadataNames, 2
  id_monitors_500, 3
  or_monitors_500, 11
  wa_airfire_meta, 28
  wa_monitors_500, 28

∗Topic environment
  getLocationDataDir, 3
  LocationDataDir, 4
  setLocationDataDir, 11

  coreMetadataNames, 2
  getLocationDataDir, 3, 4, 12
  id_monitors_500, 3, 11, 29
  location_createID, 4
  location_getSingleAddress_Photon, 5
  location_getSingleElevation_USGS, 6
  location_initialize, 7
  LocationDataDir, 3, 4, 12
  mazama_initialize, 10
  MazamaLocationUtils, 8
  MazamaLocationUtils-package
    (MazamaLocationUtils), 8
  or_monitors_500, 4, 11, 29
  setLocationDataDir, 3, 4, 11, 20
  table_addColumn, 12, 21, 24
  table_addLocation, 13, 15, 22, 25
  table_addSingleLocation, 14, 15, 22, 25
  table_export, 15
  table_getLocationID, 16
  table_getNearestDistance, 17
  table_getNearestLocation, 17
  table_getRecordIndex, 18
  table_initialize, 19
  table_load, 20
  table_removeColumn, 12, 21, 21, 24
  table_removeRecord, 14, 15, 22, 25
  table_save, 23
  table_updateColumn, 12, 24
  table_updateSingleRecord, 14, 15, 22, 25
  validateLonLat, 26
  validateLonsLats, 27
  validateMazamaSpatialUtils, 27

  wa_airfire_meta, 28
  wa_monitors_500, 4, 11, 28