Package 'EJAMejscreenapi'

February 12, 2023

Title Basic interface to EJScreen report data for each of a batch of circular buffers

Version 2.1.1

Author Mark Corrales

Maintainer Mark Corrales <corrales.mark@epa.gov>

License MIT + file LICENSE.md

URL https://github.com/USEPA/EJAMejscreenapi

Description Basic user interface for obtaining EJScreen batch results
(environmental and demographic conditions near multiple sites).
This R Shiny app provides a very basic user interface that lets a user
specify a radius in miles for circular buffering, and then
upload a csv file of point locations, with a column each for lat/lon,
and it draws a very simple map of circular buffers at those points.
It then can request from EPA servers an EJScreen standard report on each buffer.
The site-specific results, obtained one at a time (slowly) via an EJScreen API,
are compiled into a single table, one row per buffered point, with all the
indicator values as columns. Any extra columns uploaded along with lat/lon are
included as the first set of columns in the results table. The results table
can be sorted by a column, or searched (e.g., by state or facility name).
The full table can be downloaded as a csv file that can be opened as a spreadsheet.
This is intended as a very basic, slow interim tool for batches of EJScreen reports,
while the new Environmental Justice Analysis Multisite (EJAM) tool is developed.
Important caveat: These site-by-site results in many cases cannot be summarized
accurately as overall statistics on the population overall or average person.

Depends R (>= 2.10),
magrittr,
leaflet,
data.table,
shiny

Imports config (>= 0.3.1),
golem (>= 0.3.3),
pkgload,
DT,
tidyr,
readr,
ggplot2,
urltools,
httr,

jsonlite,
openxlsx,
readxl,
XML,
sp,
viridis,
leaflet.extras2

Suggests usethis,
devtools,
rsconnect,
plumber,
testthat (>= 3.0.0),
spelling

Remotes github::USEPA/EJAM,
github::USEPA/EJAMblockdata,
github::USEPA/EJAMfrsdata,
github::USEPA/EJAMbatch.summarizer,
github::USEPA/EJAMejscreendata,
github::USEPA/EJAMejscreenapi

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

LazyData true

Config/testthat/edition 3

Language en-US

R topics documented:

`addlinks_clusters_and_sort_cols` *Add Link to launch EJScreen for given point(s)*

Description

Add or update, and reorder, columns with results

Usage

```
addlinks_clusters_and_sort_cols(results_table)
```

Arguments

`results_table`	from ejscreenapi function (buffer a batch)

Details

This will Create weblinks to maps, as EJScreenMAP column, Fix weblinks to pdf-like reports, as EJScreenPDF column (from the pdfurl column), Add a column to flag sites that are close to other sites, and Put certain columns first.

Value

the input table but with extra columns

See Also

`ejscreenmaplink()` `near_eachother()`

`app_run_EJAMejscreenapi`

Launch the Shiny Application in RStudio

Description

This lets you launch the shiny web app from RStudio

Usage

```
app_run_EJAMejscreenapi(
  onStart = NULL,
  options = list(),
  enableBookmarking = NULL,
  uiPattern = "/",
  ...
)
```

Arguments

`onStart`	A function that will be called before the app is actually run. This is only needed for `shinyAppObj`, since in the `shinyAppDir` case, a `global.R` file can be used for this purpose.
`options`	Named options that should be passed to the `runApp` call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify `width` and `height` parameters which provide a hint to the embedding environment about the ideal height/width for the app.
`enableBookmarking`	Can be one of `"url"`, `"server"`, or `"disable"`. The default value, `NULL`, will respect the setting from any previous calls to `enableBookmarking()`. See `enableBookmarking()` for more information on bookmarking your app.
`uiPattern`	A regular expression that will be applied to each `GET` request to determine whether the `ui` should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
`...`	arguments to pass to golem_opts. See `?golem::get_golem_options` for more details.

Details

`app_run_EJAM()` is like `EJAM::run_app()` `app_run_EJAMejscreenapi()` is like `run_app()` `app_run_EJAMbatch.sum`
is like `EJAMbatch.summarizer::run_app()`

`boxplots_ratios` *quick boxplots of demographics across sites as ratios to US means*

Description

sdf

Usage

```
boxplots_ratios(
  x,
  selected_dvar_colname = "Demog.Index",
  selected_dvar_nicename = selected_dvar_colname,
  towhat_nicename = "US average",
  wheretext = "Near"
)
```

Arguments

`x`	data.frame that is the result of something like (ratios_to_avg(out))$ratios_d where for example out <- ejscreenapi_plus(testpoints_501:4,)
`selected_dvar_colname`	default is "Demog.Index"
`selected_dvar_nicename`	default is "Demog.Index"
`towhat_nicename`	default is "US average"
`wheretext`	Use in plot subtitle. Default is "Near" but could be "Within 5km of" for example. If it is a number, n, it will set wheretext to "Within n miles of"

Details

see notes in server code of batch.summarizer about histograms as a way to compare distributions to a benchmark, etc. This graphic is just a quick interim solution that could merge with or be replaced by the histograms in the batch sum code. See also https://r-graph-gallery.com/89-box-and-scatter-plot-with-ggplot2.html

```
    This will be revised to make shorter nicer variable names for the graphic
    names.d.nice.api should be looked up based on selected_dvar_colname
    dnames_expected <- c(
      "Demog.Index", "pctmin", "pctlowinc", "pctlths","pctlingiso",
      "pctunder5", "pctover64", "pctunemployed")
    dnice <- c(
      "Demog.Ind.", "
       "
    if (names(x) == dnames_expected) {nicenames <- dnice}
```

```
May want to compare to boxplots of complete nationwide range of indicator values.

  y <- EJAM::blockgroupstats[ , ..names_d]
 y <- scale(y,  sapply(y, FUN=function(x) {Hmisc::wtd.mean(x, wts, na.rm=na.rm)}), center=FALSE)
  y <- as.data.frame(y)


To communicate whether this is skewed to the right (more high scores than might expect)
 also could say that
 X
 e.g., 20
 (which is more/less than one might expect - leaving aside statistical significance
 ie whether this could be by chance if sites were randomly picked
 from US block groups or people's bg scores)
```

Examples

```
 # see source code of [ejscreenapi_script()]
 myrad <- mean(testoutput_ejscreenapi_plus_50$radius.miles)
boxplots_ratios(ratios_to_avg(testoutput_ejscreenapi_plus_50)$ratios_d, wheretext=myrad)
```

`convert_units` *Convert units of distance or area*

Description

`convert_units` converts distance/length or area from specified units to other specified units.

Usage

```
convert_units(x, from = "km", towhat = "mi")
```

Arguments

`x`	A number or vector of numbers to be converted.
`from`	A string specifying original units of input parameter. Default is 'km' which is kilometers. Note all must be in the same units. Units can be specified as any of the following: c('millimeter', 'millimeters', 'centimeter', 'centimeters', 'meter', 'meters', 'kilometer', 'kilometers', "mm", "cm", "m", "km", "sqmm", "sqcm", "sqm", "sqkm", "mm2", "cm2", "m2", "km2", 'inch', 'inches', 'foot', 'feet', 'yard', 'yards', 'mile', 'miles', "in", "ft", "yd", "mi", "sqin", "sqft", "sqyd", "sqmi", "in2", 'ft2', 'yd2', 'mi2') Note that m2 is for square meters not square miles.
`towhat`	A strings specifying new units to convert to. Default is 'mi' which is miles.

Details

This function takes a number, or vector of numbers, representing distance/length or area in one type of specified units, such as miles, and returns the corresponding number(s) converted to some other units, such as kilometers. Units can be specified in various ways. All inputs must be in the same units. All outputs must be in a single set of units as well.

NOTE: For some purposes, Census Bureau does this: "The ANSI standard for converting square kilometers into square miles was used (1 square mile = 2.58998811 square kilometers)." (see `https://www.census.gov/geo/reference/state-area.html`) but the conversions in this function use 2.5899881034 not 2.58998811 sqkm/sqmi. The difference is only 6.6 per billion (roughly 1 in 152 million), which is less than one tenth of a square kilometer out the entire USA.

Value

Returns a number or vector of numbers then length of the input x, with each element corresponding to an input element converted to new units.

See Also

`get.distances` which allows you to specify a search radius and get distances only within that radius, and related functions.

Examples

```
convert_units(1, 'mi', 'km')
convert_units(c(1e6, 1), 'sqm', 'sqkm')
```

`default_points_shown_at_startup`
set of locations shown by default at startup

Description

set of locations shown by default at startup

`echo_colids_from_num_name_group`
Interpret and check the list of requested columns to be asked of the ECHO API get_facility_info

---

Description

Interpret and check the list of requested columns to be asked of the ECHO API get_facility_info

Usage

```
echo_colids_from_num_name_group(x = NULL)
```

Arguments

x	vector of specifiers of variables needed to get from the ECHO facility query API, which can be the variable column id that the API needs (1 through about 316), or the actual ObjectName (variable name) found in varsinfo_ECHO_API, or a word spefying a group of variables like critical or best or others that are logical class columns in varsinfo_ECHO_API or the word all (for all available, about 316).

Value

Not the same sort order or length as input necessarily! A valid list of numbers that are the ColumnID numbers

See Also

`get_facility_info_via_ECHO()`

Examples

```
x =echo_colids_from_num_name_group(c('critical', 'NC', 'CensusBlockGroup' ))
x
varsinfo_ECHO_API$ObjectName[match( x, varsinfo_ECHO_API$ColumnID)]
echo_colids_from_num_name_group(300:400)
echo_colids_from_num_name_group(5:1)
echo_colids_from_num_name_group(c(5:1,1:3))
```

`EJAMejscreenapi` *ejscreenapi tool for the Environmental Justice (EJ) Analysis Multisite tool*

Description

Basic user interface for obtaining EJScreen batch results (environmental and demographic conditions near multiple sites). This R Shiny app provides a very basic user interface that lets a user specify a radius in miles for circular buffering, and then upload a csv file of point locations, with a column each for lat/lon, and it draws a very simple map of circular buffers at those points. It then can request from EPA servers an EJScreen standard report on each buffer. The site-specific results, obtained one at a time (slowly) via an EJScreen API, are compiled into a single table, one row per buffered point, with all the indicator values as columns. Any extra columns uploaded along with lat/lon are included as the first set of columns in the results table. The results table can be sorted by a column, or searched (e.g., by state or facility name). The full table can be downloaded as a csv file that can be opened as a spreadsheet. This is intended as a very basic, slow interim tool for batches of EJScreen reports, while the new Environmental Justice Analysis Multisite (EJAM) tool is developed. Important caveat: These site-by-site results in many cases cannot be summarized accurately as overall statistics on the population overall or average person.

Key Functions:

- `runApp()` Launch the Shiny app (web interface). See more at help('EJAMejscreenapi')
- `ejscreenapi_script()` In RStudio console or in a script, get summary results without the shiny app web interface x <- ejscreenapi_script('./inst/testdata/testpoints_05.csv')

See Also

Useful links:

- `https://github.com/USEPA/EJAMejscreenapi`

`ejscreenapi` *Use EJScreen API to get stats on multiple circular buffers*

Description

Get a data.table of EJScreen report results for multiple circular buffers.

Usage

```
ejscreenapi(
  lon,
  lat,
  radius = 3,
  unit = "miles",
  wkid = 4326,
  report_every_n = 1000,
  save_when_report = FALSE,
  format_report_or_json = "pjson",
  on_server_so_dont_save_files = FALSE,
  ipurl = "ejscreen.epa.gov",
  updateProgress = NULL
)
```

Arguments

`lon`	Longitude numeric vector
`lat`	Latitude numeric vector
`radius`	radius of circular buffer
`unit`	miles (default) or kilometers
`wkid`	optional spatial reference code
`report_every_n`	Should it report ETA snd possibly save interim file after every n points
`save_when_report`	optional, write .rdata file to working directory with results so far, after ever n points, to have most results even if it crashes
`format_report_or_json`	Not implemented. default is pjson but could modify to allow it to be report to get a pdf
`on_server_so_dont_save_files`	FALSE by default, but TRUE prevents saving any progress or crash-related files
`ipurl`	which URL or IP to try
`updateProgress`	Used to create progress bar in Shiny app

Details

This relies on `ejscreenapi1()` to request URL of pdf report on each site via the API, and does some error checking, but like `ejscreenapi1()` it does a GET request via API and then parses the JSON results from the GET request, cleans it up, adds URLs as links, compiles it as a data.table, enables a progress bar, etc.

Specify a radius and vector of latitude longitude points, and get for a buffer the population weighted mean value of each raw indicator like percent low-income, and total population count, and percentiles for those raw indicator scores, all from EJScreen, as in an EJScreen standard report. Note that this API is fairly slow, so it is fine for 10 sites, but not large numbers. It does maybe about 7k to 10k sites per hour, for circular buffers of 1 or 3 mile radius. It sometimes crashes, with a JSON lexical error, which may be caused by unreliable results from the API rather than the code requesting results via the API. See `'https://www.epa.gov/ejscreen/ejscreen-api'`

Examples

```
 ## Not run:
 # Specify size of buffer circle and pick random points as example data
 myradius <- 3
 pts <- structure(list(lon = c(-96.4798957, -111.7674343, -75.4173589,
 -95.9573172, -87.8402677, -77.9996191, -73.920702, -79.9545638,
 -76.0638877, -114.9881473), lat = c(31.782716, 33.7522735, 39.8697972,
 33.2522474, 41.9763992, 38.4661259, 41.2940801, 32.8099327, 40.9888266,
 36.0043628), id = 1:10), row.names = c(NA, -10L), class = "data.frame")

  out <- ejscreenapi(pts$lon, lat=pts$lat, radius = myradius)
 t(out[1:2,])

## End(Not run)
```

`ejscreenapi1` *Use EJScreen API to get stats on ONLY ONE circular buffer*

Description

Get EJScreen report results for one circular buffer, as a data.frame

Usage

```
ejscreenapi1(
  lon,
  lat,
  radius = 3,
  unit = "miles",
  wkid = 4326,
  format_report_or_json = "pjson",
  ipurl = "ejscreen.epa.gov"
)
```

Arguments

`lon`	Longitude numeric vector
`lat`	Latitude numeric vector
`radius`	radius, in miles, of circular buffer
`unit`	miles (default) or kilometers
`wkid`	optional spatial reference code
`format_report_or_json`	Not implemented. default is pjson but could modify to allow it to be report to get a pdf
`ipurl`	IP or URL start

Details

Relies on `ejscreenRESTbroker()` for the actual request via API. Specify a radius and vector of latitude longitude points, and get for a buffer the population weighted mean value of each raw indicator like percent low-income, and total population count, and percentiles for those raw indicator scores, all from EJScreen, as in an EJScreen standard report. Note that this API is fairly slow, so it is fine for 10 sites, but not large numbers. See `'https://www.epa.gov/ejscreen/ejscreen-api'`

See Also

`ejscreenapi_script()` `ejscreenapi_plus()` `ejscreenapi()` that uses `ejscreenapi1()` and `ejscreenRESTbroker()`

Examples

```
 ## Not run:
 # Specify size of buffer circle and pick random points as example data
 myradius <- 3
 pts <- structure(list(lon = c(-96.4798957, -111.7674343, -75.4173589,
 -95.9573172, -87.8402677, -77.9996191, -73.920702, -79.9545638,
 -76.0638877, -114.9881473), lat = c(31.782716, 33.7522735, 39.8697972,
 33.2522474, 41.9763992, 38.4661259, 41.2940801, 32.8099327, 40.9888266,
 36.0043628), id = 1:10), row.names = c(NA, -10L), class = "data.frame")

  out <- ejscreenapi(pts$lon, lat=pts$lat, radius = myradius)

 t(out[1:2,])

## End(Not run)
```

---

ejscreenapi_plus — *Given a set of points (lat lon), get table of EJScreen API results near each*

Description

Using EJScreen API without Shiny app interface

Usage

```
ejscreenapi_plus(
  x,
  y = NULL,
  radius = 3,
  unit = "miles",
  wkid = 4326,
  report_every_n = 1000,
  save_when_report = FALSE,
  format_report_or_json = "pjson",
  on_server_so_dont_save_files = FALSE,
  ipurl = "ejscreen.epa.gov",
  mapping_for_names = EJAMejscreenapi::map_headernames,
  usewhichnames = "friendly"
)
```

Arguments

`x`	longitudes; or path/filename to xlsx or csv with lat, lon; or data.frame or data.table with lat,lon
`y`	latitudes, or ignored if x was a file or table with lat,lon info.
`radius`	circular buffer radius (in miles by default, unless unit changed)
`unit`	default is miles
`wkid`	do not use
`report_every_n`	default is to provide an update every so often
`save_when_report`	default is FALSE but if TRUE it saves work in progress every so often
`format_report_or_json`	do not use
`on_server_so_dont_save_files`	default is FALSE, but set to TRUE if this is run on a server
`ipurl`	change only if different URL has to be used for the EJScreen API
`mapping_for_names`	a table that translates between original (as on FTP site), short friendly (useful in coding or analysis), and long complete variable names (for clearer Excel headers). This can be read from a csv file or from data in a package. Not documented here, as format may change.
`usewhichnames`	default is to use the short friendly ones

Details

See `ejscreenapi_script()` for more details on this.

```
  [ejscreenapi_script()] uses functions below, but returns a list with table, map, plot, etc.
     [ejscreen_plus()] accepts file or table or vectors of point data,
       uses [ejscreenapi()] to get EJScreen stats,
       and then prepends input table and renames columns, to return a table.
     [ejscreenapi()]   gets EJ stats for many points as a data.table of many rows.
       [ejscreenapi1()] gets EJ stats for 1 point via API, as data.frame of 1 row.
         [ejscreenRESTbroker()] gets EJ stats for one point as JSON.
```

Value

Returns a data.frame of results, one row per buffer (site), one column per indicator, with roughly 200 columns. `ejscreenapi_plus()` returns that as one element of a list that also has a map and plot.

See Also

`ejscreenapi_script()` which also demonstrates a map and a plot, and accepts filename as input pts. see `ejscreenapi()` that uses `ejscreenapi1()` and `ejscreenRESTbroker()`

Examples

```
 # see [ejscreenapi_script()] for examples
  pts <- data.frame(
   siteid = 1:2,
   sitename = c("example site A", "example site B"),
   lon = c(-91.132107, -91.09),
   lat = c(30.494982,   30.45)
  )
  myradius <- 1
  x <- testoutput_ejscreenapi_plus_50
  # pts2 <-  system.file("testdata/Sample12.xlsx",  package="EJAMejscreenapi")
  # x <- ejscreenapi_plus(pts2, radius = myradius, usewhichnames = "long")
  # x <- ejscreenapi_plus(pts$lon, pts$lat, radius = myradius, usewhichnames = "long",)
  # x <- ejscreenapi_plus(pts,  radius = myradius, usewhichnames = "long")
  t(x[1,3:ncol(x)])
  DT::datatable(x[ , 3:ncol(x)])
 names(x) <- fixnames_to_type(names(x), oldtype="longname_tableheader", newtype="newnames_ejscreenapi")
  boxplots_ratios(ratios_to_avg(x), wheretext = myradius)
  # look at the weblinks in RStudio viewer window
  html_print(HTML(paste( paste(x[,1], collapse="<br>"),
   "<p></p>", paste(x[,2], collapse="<br>"))))
```

---

ejscreenapi_script — *Given a list of points, get table of EJScreen API results near each + map + boxplots*

Description

Using EJScreen API without Shiny app interface

Usage

```
ejscreenapi_script(
  x,
  y = NULL,
  radius = 3,
  maxradiusmiles = 10,
  save_map = TRUE,
  see_map = TRUE,
  save_plot = TRUE,
  see_plot = TRUE,
  save_table = TRUE,
  see_table = FALSE,
  folder = getwd(),
  ...
)
```

Arguments

`x`	longitudes; or path/filename to xlsx or csv with lat, lon; or data.frame or data.table with lat,lon
`y`	latitudes, or ignored if x was a file or table with lat,lon info.
`radius`	in miles - gets data on all residents within that distance from each point
`maxradiusmiles`	optional in case you want to use a radius of more than default cap
`save_map`	logical, whether to save png image file locally
`see_map`	logical, whether to display interactive map
`save_plot`	logical, whether to save png image file locally
`see_plot`	logical, whether to display plot (boxplots)
`save_table`	logical, whether to save table of data in a file locally
`see_table`	logical, whether to display interactive table
`folder`	full path of directory in which to save files like map, plot, table Default is working directory.
`...`	passed to ejscreenapi_plus()

Details

```
 See also help for [ejscreenapi_plus()].

 This is one function that accepts a data.table of points (columns lat, lon)
 or excel file that has that info,
 and radius, and returns a list of results including a
 table of scores that are like the EJScreen
 standard buffer report, but with one row per site (input point).

This requests each site report via the EJScreen API and [ejscreenapi_plus()] to get
 buffer results summarizing environmental and demographic indicator scores
 for the average person within a specified distance of each point.
 It returns a list of 4 items: table, map, plot, us.ratios

It uses at least these: [ejscreenapi_plus()], [ejscreenapi()], [addlinks_clusters_and_sort_cols()
[fixcolnames()], [linkify()], [meters_per_mile] constant, [map_headernames] table, [make.popups.a
 [ratios_to_avg()], [boxplots_ratios()]

 [ejscreenapi_script()] uses functions below and returns a list with table, map, plot, etc.
   [ejscreen_plus()] accepts file of point data, uses [ejscreenapi()] to get EJScreen stats,
       and then prepends input table and renames columns.
     [ejscreenapi()]   gets EJ stats for many points as a data.table of many rows.
       [ejscreenapi1()] gets EJ stats for 1 point via API, as data.frame of 1 row.
         [ejscreenRESTbroker()] gets EJ stats for one point as JSON.
```

Value

Returns a list with map, boxplot, table, us.ratios:

- map of sites with popups of EJ stats, as returned plot, viewed interactive map and .png
- graphic boxplot of some demographics as ratios to average, as returned plot, viewed noninteractive plot and .png
- table of results as a returned data.frame, viewed interactive datatable, and .xlsx and .csv
- us.ratios is a data.frame of one row per site and one column per indicator as ratios to US average.

See Also

ejscreenapi_plus()

Examples

```
## Not run:
 pts <- testpoints_50[1:3, ] # sample data from package
 mapfast(pts)
 #pts <- system.file("testdata/testpoints_05.csv",package="EJAMejscreenapi")
 #pts <- system.file("testdata/Sample12.xlsx",    package="EJAMejscreenapi")

 x <- ejscreenapi_script(pts,
   save_map = F, save_plot = F, save_table = F, folder = "~", see_table = T)

 myradius <- 1 # in miles
 # myradius <- 5000 / meters_per_mile # 5 kilometer radius, approx 3.1 miles
 # Get results from server by using API
 x <- ejscreenapi_script(
   pts=pts, radius = myradius,
   save_table = FALSE, save_map = FALSE, save_plot = FALSE)

 # see format of output results
 names(x) # [1] "table" "map"   "plot"

 # For this table view, remove map and pdf URL columns
 y <- x$table[ , !grepl("EJScreen", names(x$table))]
 t(y[1, ])  # see one column of results
 DT::datatable(y)  # see interactive data table view in viewer of RStudio

 # View links in RStudio Viewer window rather than shiny app or Excel
 html_print(HTML(paste(paste(x$table[,"EJScreenPDF"], collapse="<br>"),
    "<p></p>", paste(x$table[,"EJScreenMAP"], collapse="<br>") )  ))

 # View the boxplots of results
 x$plot

 # Save the plot as a file
 png(filename = file.path(folder,"Boxplot of EJ stats.png"),
   width = 1200, height = 600)
 x$plot
 dev.off()
 x$map
 x$map %>% leaflet.extras2::easyprintMap(
   filename = file.path(folder, "map.png"), exportOnly = TRUE)
```

```
## End(Not run)
```

`ejscreenmaplink` *provides URL for EJScreen map centered at given point*

Description

provides URL for EJScreen map centered at given point

Usage

```
ejscreenmaplink(where)
```

Arguments

`where`	string containing lat lon of 1 point as a comma separated string in this format: 30.450000,-91.090000

Value

URL

`ejscreenRESTbroker` *Use EJScreen API for one circular buffer, get raw json or report output*

Description

Use EJScreen API

Usage

```
ejscreenRESTbroker(
  lon,
  lat,
  url = "https://ejscreen.epa.gov/mapper/ejscreenRESTbroker.aspx?namestr=",
  wkid = 4326,
  distance = 1,
  unit = 9035,
  f = "pjson",
  ipurl = "ejscreen.epa.gov"
)
```

Arguments

`lon`	a longitude
`lat`	a latitude
`url`	URL base for API
`wkid`	spatial reference
`distance`	radius of the circular buffer
`unit`	9035 for miles, 9036 for kilometers
`f`	pjson for JSON, report for pdf report
`ipurl`	fixed ip or domain/URL to try

Details

Note the public IP is 204.47.252.51 and internal is different. See https://www.epa.gov/ejscreen/ejscreen-api

See Also

`ejscreenapi_script()` `ejscreenapi_plus()` `ejscreenapi()` that uses `ejscreenapi1()` and `ejscreenRESTbroker()`

Examples

```
## Not run:
  browseURL(ejscreenRESTbroker(lon = -80, lat = 42, f = 'report'))
  x = (ejscreenRESTbroker(lon = -80, lat = 42))
  names(jsonlite::fromJSON(rawToChar(x$content)))

## End(Not run)
```

`expand.gridMatrix` *Similar to expand.grid, but returns a matrix not data.frame*

Description

This function is similar to `expand.grid`, in the sense that it returns a matrix that has 2 columns, one for each input, and one row per combination, cycling through the first field first. It differs from expand.grid in that this returns a matrix not data.frame, only accepts two parameters creating two columns, for now, and lacks the other parameters of expand.grid

Usage

```
## S3 method for class 'gridMatrix'
expand(x, y)
```

Arguments

`x`	required vector
`y`	required vector

Value

This function returns a matrix and tries to assign colnames based on the two input parameters. If they are variables, it uses those names as colnames. Otherwise it uses "x" and "y" as colnames.

See Also

`expand.grid`

Examples

```
expand.gridMatrix(99:103, 1:2)
zz <- 1:10; top <- 1:2
expand.gridMatrix(zz, top)
```

`fixcolnames` *helper function to rename variables that are colnames of data.frame Changes column names to friendly or long from original names in FTP site file*

Description

helper function to rename variables that are colnames of data.frame Changes column names to friendly or long from original names in FTP site file

Usage

```
fixcolnames(
  names_table_as_displayed,
  towhichnames = "friendly",
  fromwhichnames = "original",
  mapping_for_names
)
```

Arguments

`names_table_as_displayed`	vector of colnames
`towhichnames`	friendly or long or original
`fromwhichnames`	friendly or long or original
`mapping_for_names`	default is a dataset already in the package. A data.frame passed to `fixnames()` to do the work with columns oldnames (original), longname_tableheader (long), newnames_ejscreenapi (friendly)

Value

Vector or new column names same length as input

See Also

`fixnames_to_type()` `fixcolnames()` `fixnames()`

Examples

```
 # tests
test.original <- c("S_E_TRAFFIC_PER","RAW_D_INCOME","N_P_PM25","N_E_CANCER",
 "unfound", NA)
test.friendly <- c(
  "state.pctile.traffic.score", "pctlowinc", 'pctile.EJ.DISPARITY.pm.eo',
  'us.avg.cancer',"unfound", NA)
test.long <- c(
  paste0("State percentile for Traffic Proximity and Volume",
   " (daily traffic count/distance to road)"),
  "Raw data for Low Income Population",
  "National percentile for EJ Index for Particulate Matter (PM 2.5)",
 "National average for NATA Air Toxics Cancer Risk (risk per MM)","unfound",NA)
# cbind(test.original, test.friendly, test.long)

fixcolnames(test.original,
  fromwhichnames='original', towhichnames='friendly') == test.friendly
fixcolnames(test.original,
  fromwhichnames='original', towhichnames='long')     == test.long
  # from=to, so just returns unchanged:
fixcolnames(test.original,
  fromwhichnames='original', towhichnames='original') == test.original
  # wrong from, so just returns unchanged:
fixcolnames(test.original,
  fromwhichnames='long',     towhichnames='original') == test.original
# fixcolnames(test.original,
  fromwhichnames='wrong',  towhichnames='original')  # fails

 # from=to, so just returns unchanged:
fixcolnames(test.friendly,
  fromwhichnames='friendly', towhichnames='friendly') == test.friendly
fixcolnames(test.friendly,
  fromwhichnames='friendly', towhichnames='long')     == test.long
fixcolnames(test.friendly,
  fromwhichnames='friendly', towhichnames='original') == test.original
  # wrong from, so just returns unchanged:
fixcolnames(test.friendly,
  fromwhichnames='long', towhichnames='original')     == test.friendly
# fixcolnames(test.friendly,
  fromwhichnames='wrong', towhichnames='original')  # fails

fixcolnames(test.long,
  fromwhichnames = 'long', towhichnames = 'friendly') == test.friendly
 # from=to, so just returns unchanged:
fixcolnames(test.long,
  fromwhichnames = 'long', towhichnames = 'long')     == test.long
fixcolnames(test.long,
  fromwhichnames = 'long', towhichnames = 'original') == test.original
 # wrong from, so just returns unchanged:
fixcolnames(test.long,
  fromwhichnames = 'friendly', towhichnames = 'original') == test.long
# fixcolnames(test.long,
  fromwhichnames = 'wrong', towhichnames = 'original')  # fails
```

fixnames — *helper function to rename variables that are colnames of data.frame Changes column names to friendly from original names in FTP site file*

Description

helper function to rename variables that are colnames of data.frame Changes column names to friendly from original names in FTP site file

Usage

```
fixnames(headernames, mapping_for_names)
```

Arguments

`headernames`	vector of colnames
`mapping_for_names`	data.frame passed to `fixnames()` to do the work with columns oldnames (original), longname_tableheader (long), newnames_ejscreenapi (friendly)

Value

Vector or new column names same length as input

See Also

`fixcolnames()` `fixnames()` `fixnames_to_type()`

fixnames_to_type — *helper function to change elements of namesnow from an oldtype to a newtype of names*

Description

helps convert between original, friendly, and long versions of variable names

Usage

```
fixnames_to_type(namesnow, oldtype, newtype, mapping_for_names)
```

Arguments

`namesnow`	vector of strings, such as from colnames(x)
`oldtype`	string with name of a column in data.frame mapping_for_names, and that column has old column names that overlap with those in namesnow
`newtype`	string with name of a column in data.frame mapping_for_names, and that column has old column names that overlap with those in namesnow
`mapping_for_names`	data.frame passed to `fixnames()` to do the work with colnames that are referred to by oldtype and newtype

Details

using lookup table mapping_for_names, finds each namesnow in the column specified by oldtype and replaces it with the corresponding string in the column specified by newtype

Value

Vector or new column names same length as input

See Also

`fixnames_to_type()` `fixcolnames()` `fixnames()`

`get.distances.all` *Find all distances between two sets of points (based on lat/lon)*

Description

Returns all the distances from one set of geographic points to another set of points. Can return a matrix of distances (m x n points) or vector or data.frame with one row per pair. Lets you specify units and whether you need lat/lon etc, but essentially just a wrapper for the **sp** package for the `spDistsN1` and `SpatialPoints` functions.

Usage

```
get.distances.all(
  frompoints,
  topoints,
  units = "miles",
  return.crosstab = FALSE,
  return.rownums = TRUE,
  return.latlons = TRUE,
  as.df = TRUE
)
```

Arguments

`frompoints`	A matrix or data.frame with two cols, 'lat' and 'lon' with datum=WGS84 assumed.
`topoints`	A matrix or data.frame with two cols, 'lat' and 'lon' with datum=WGS84 assumed.
`units`	A string that is 'miles' by default, or 'km' for kilometers, specifying units for distances returned.
`return.crosstab`	Logical value, FALSE by default. If TRUE, value returned is a matrix of the distances, with a row per frompoint and col per topoint.
`return.rownums`	Logical value, TRUE by default. If TRUE, value returned also includes two extra columns: a col of index numbers starting at 1 specifying the frompoint and a similar col specifying the topoint. If crosstab=TRUE, ignores return.rownums and return.latlons

`return.latlons`	Logical value, TRUE by default. If TRUE, value returned also includes four extra columns, showing fromlat, fromlon, tolat, tolon. If crosstab=TRUE, ignores return.rownums and return.latlons
`as.df`	Logical, default is TRUE, in which case returns a data.frame (unless vector), otherwise a matrix (unless vector).

Details

```
  *** Probably slower than it needs to be partly by using data.frame
   instead of matrix class? Maybe 10-20
  Just using get.distances.all is reasonably fast? (30-40 seconds for
     100 million distances, but slow working with results so large),
 Sys.time(); x=get.distances.all(testpoints(1e5), testpoints(1000),
   return.crosstab=TRUE); Sys.time()
     "2015-03-10 18:59:08 EDT"
     "2015-03-10 18:59:31 EDT"  23 SECONDS  for 100 million distances
       IF NO PROCESSING OTHER THAN CROSSTAB
 Sys.time(); x=get.distances.all(testpoints(1e6), testpoints(100),
    return.crosstab=TRUE); Sys.time()
     "2015-03-10 21:54:11 EDT"
     "2015-03-10 21:54:34 EDT"  23 SECONDS for 100 million distances
       (1m x 100, or 100k x 1000)
 Sys.time(); x=get.distances.all(testpoints(1e6), testpoints(300),
   return.crosstab=TRUE); Sys.time()
     "2015-03-10 21:56:11 EDT"
     "2015-03-10 21:57:18 EDT"  67 seconds for 300 million pairs.
  plus 20 seconds or so for x[x>100] <- Inf
       # so 11m blocks to 1k points could take >40 minutes!
       (you would want to more quickly remove the ones outside some radius)
           >3 minutes per 100 sites?
           About 2.6 seconds per site for 11m blocks?

    Sys.time(); x=get.distances.all(testpoints(1e5), testpoints(1000),
      units='miles',return.rownums=TRUE); Sys.time()
    "2015-03-09 21:23:04 EDT"
    "2015-03-09 21:23:40 EDT"  36 SECONDS IF DATA.FRAME ETC. DONE
      TO FORMAT RESULTS AND GET ROWNUMS
   Sys.time(); x=get.distances.all(testpoints(1e5), testpoints(1000),
     units='miles',return.rownums=TRUE)$d; Sys.time()
    "2015-03-09 21:18:47 EDT"
    "2015-03-09 21:19:26 EDT" 49 SECONDS IF DATA.FRAME ETC. DONE
      TO FORMAT RESULTS AND GET ROWNUMS IN get.distances.all
```

Value

By default, returns a dataframe that has 3 columns: fromrow, torow, distance (where fromrow or torow is the row number of the corresponding input, starting at 1). If return.crosstab=FALSE, which is default, and return.rownums and/or return.latlons is TRUE, returns a row per from-to pair, and columns depending on parameters, sorted first cycling through all topoints for first frompoint, and so on. If return.crosstab=FALSE and return.rownums and return.latlons are FALSE, returns a vector of distances in same order as rows described above. If return.crosstab=TRUE, returns a matrix of distances, with one row per frompoint and one column per topoint.

See Also

latlon_infer() get.distances() which allows you to specify a search radius and get distances only within that radius which can be faster, get.distances.prepaired() for finding distances when data are already formatted as pairs of points, get.nearest() which finds the distance to the single nearest point within a specified search radius instead of all topoints, and proxistat or proxistat2() which will which create a proximity score for each spatial unit based on distances to nearby points.

Examples

```
set.seed(999)
t1=testpoints(1)
t10=testpoints(10)
t100=testpoints(100, minlat=25,maxlat=48)
t1k=testpoints(1e3)
t10k=testpoints(1e4)
t100k=testpoints(1e5)
t1m=testpoints(1e6)
#t10m=testpoints(1e7)

get.distances.all(t1, t1)
get.distances.all(t1, t10[2, , drop=FALSE])
x=get.distances.all(t10, t100[1:20 , ], units='km')
 plot(x$tolon, x$tolat,pch='.')
 points(x$fromlon, x$fromlat)
 with(x, linesegments(fromlon, fromlat, tolon, tolat ))
 with(x[x$d<500, ], linesegments(fromlon, fromlat, tolon, tolat ,col='red'))
x=get.distances.all(t10, t1k); head(x);summary(x$d)
x=get.distances.all(t10, t1k, units='km'); head(x);summary(x$d)
x=get.distances.all(t10, t1k, units='km'); head(x);summary(x$d)

## Not run:
require(UScensus2010blocks) # for the get.blocks() function and dataset
blocks <- get.blocks(fields=c('fips','lat','lon'),charfips = FALSE)


## End(Not run)

   test.from <- structure(list(fromlat = c(38.9567309094, 45),
     fromlon = c(-77.0896572305, -100)), .Names = c("lat", "lon"),
     row.names = c("1", "2"), class = "data.frame")

   test.to <- structure(list(tolat = c(38.9575019287, 38.9507043428, 45),
    tolon = c(-77.0892818598, -77.2, -90)),
    .Names = c("lat", "lon"), class = "data.frame",
    row.names = c("1", "2", "3"))
 test.to.NA = rbind(c(NA,NA), test.to[2:3,])
 test.from.NA = rbind(test.from[1,], c(NA,NA))

get.distances.all(test.from, test.to)
get.distances.all(test.from, test.to, return.crosstab=TRUE)
get.distances.all(test.from, test.to, return.rownums=FALSE)
get.distances.all(test.from, test.to, return.latlons=FALSE)
get.distances.all(test.from, test.to, return.latlons=FALSE,
  return.rownums=FALSE)
```

```
     # test cases
get.distances.all(test.from,    test.to.NA)
get.distances.all(test.from.NA, test.to)
get.distances.all(test.from.NA, test.to.NA)
get.distances.all(test.from[1,],test.to[1,],return.rownums=F,return.latlons=F)
get.distances.all(test.from[1,],test.to[1,],return.rownums=FALSE,return.latlons=TRUE)
get.distances.all(test.from[1,],test.to[1,],return.rownums=TRUE,return.latlons=FALSE)
get.distances.all(test.from[1,],test.to[1,],return.rownums=TRUE,return.latlons=TRUE)

get.distances.all(test.from[1,],test.to[1:3,],return.rownums=F,return.latlons=F)
get.distances.all(test.from[1,],test.to[1:3,],return.rownums=FALSE,return.latlons=TRUE)
get.distances.all(test.from[1,],test.to[1:3,],return.rownums=TRUE,return.latlons=FALSE)
get.distances.all(test.from[1,],test.to[1:3,],return.rownums=TRUE,return.latlons=TRUE)

get.distances.all(test.from[1:2,],test.to[1,],return.rownums=F,return.latlons=F)
get.distances.all(test.from[1:2,],test.to[1,],return.rownums=FALSE,return.latlons=TRUE)
get.distances.all(test.from[1:2,],test.to[1,],return.rownums=TRUE,return.latlons=FALSE)
get.distances.all(test.from[1:2,],test.to[1,],return.rownums=TRUE,return.latlons=TRUE)

round(get.distances.all(test.from[1:2,],test.to[1:3,],return.rownums=F,return.latlons=F),1)
get.distances.all(test.from[1:2,],test.to[1:3,],return.rownums=FALSE,return.latlons=T)
get.distances.all(test.from[1:2,],test.to[1:3,],return.rownums=TRUE,return.latlons=F)
get.distances.all(test.from[1:2,],test.to[1:3,],return.rownums=TRUE,return.latlons=TRUE)
get.distances.all(test.from[1:2,],test.to[1:3,], return.rownums=TRUE,
  return.latlons=TRUE, units='km')
get.distances.all(test.from[1:2,],test.to[1:3,], return.rownums=TRUE,
  return.latlons=TRUE, units='miles')

get.distances.all(test.from[1,],test.to[1:3, ], return.crosstab=TRUE)
get.distances.all(test.from[1:2,],test.to[1, ], return.crosstab=TRUE)
round(get.distances.all(test.from[1:2,],test.to[1:3, ],return.crosstab=TRUE, units='miles'),2)
round(get.distances.all(test.from[1:2,],test.to[1:3, ],return.crosstab=TRUE, units='km'),2)
```

---

get_facility_info_via_ECHO

Obsolete? Get info on EPA-regulated facilities via EPA ECHO API - query by NAICS etc.

Description

Lets you query by NAICS relatively easily. But for other queries, see `EJAMfrsdata::get_siteid_from_naics()` and `EJAMfrsdata::get_latlon_from_siteid()` etc or locate_by_id() to use FRS API, or other code in EJAM package that might make this function obsolete.

Usage

```
get_facility_info_via_ECHO(
  p_ncs = NULL,
  qcolumns = c(16, 17),
  output = "JSON",
  otherparameters = NULL,
  url_not_query = TRUE,
  testing = FALSE,
  getcsv = FALSE
)
```

Arguments

`p_ncs`	NAICS industrial code
`qcolumns`	vector specifying which variables to return (see varsinfo_ECHO_API). Column numbers work and are what the ECHO API expects, but here you can instead - or in addition - also use these words referring to groups of variables defined in this package: critical best useful programid ej to specify variables where, e.g., the word critical would get variables where varsinfo_ECHO_API$critical == TRUE
`output`	JSON by default, to get output in that format
`otherparameters`	appended text at end of URL
`url_not_query`	logical, just return the URL but not query
`testing`	logical
`getcsv`	logical, use get_download

Details

See info about NAICS industry codes at `https://www.naics.com/search`

See info about ECHO web services at `https://echo.epa.gov/tools/web-services`. Use the Metadata service endpoint for a list of available output objects, their Column Ids, and their definitions to help you build your customized output, and see examples at `https://echo.epa.gov/tools/web-services/facility-search-all-data#/Metadata` and `https://echodata.epa.gov/echo/echo_rest_services.metadata?output=JSON`

In ECHO, one can search for facilities or permits by EPA Registry ID (i.e., FRS ID) or by the Program System ID (CWA, CAA, SDWA, or RCRA). The web interface at https://echo.epa.gov/facilities/facility-search allows data entry of up to 2,000 IDs pasted from spreadsheet column, or comma- or return-separated.

Value

Tries to return a table via data.table::as.data.table(), with these columns: "ObjectId" "FacName" "RegistryID" "FacLat" "FacLong" "lat" "lon" "registry_id"

See Also

varsinfo_ECHO_API EJAMfrsdata::get_siteid_from_naics()

Examples

```
## Not run:
# mynaics <- EJAM::naics_find("petrochemical manufact")[1]  # over 1,100 facilities
mynaics <- EJAM::naics_find("Evaporated Dairy") # over 300
myvariablenumbers <- varsinfo_ECHO_API$ColumnID[varsinfo_ECHO_API$critical]
x <-  get_facility_info_via_ECHO(
 mynaics,
 qcolumns = myvariablenumbers,
 url_not_query = T # F
)
```

```
x
mapfast(x)

## End(Not run)
```

latlon2csv

helper function - combine lat/lon values into csv format Combines a vector of latitudes and a vector of longitudes into one vector of comma-separated pairs like latitude,longitude

Description

helper function - combine lat/lon values into csv format Combines a vector of latitudes and a vector of longitudes into one vector of comma-separated pairs like latitude,longitude

Usage

```
latlon2csv(lat, lon)
```

Arguments

`lat`	vector of latitudes
`lon`	vector of longitudes

Value

vector of comma-separated pairs

Examples

```
# lat_example = c(30.01,30.26,30.51)
# lon_example = c(-90.61,-90.95,-91.23)
# latloncsv_example = latlon2csv(lat_example,lon_example)
# latloncsv_example == c("30.01,-90.61", "30.26,-90.95", "30.51,-91.23")
```

latlon2nexus

helper function - combine lat/lon values to paste into NEXUS tool Converts 2 vector of values for latitude and longitude into a format you can paste into NEXUS tool lat/lon site selection box

Description

helper function - combine lat/lon values to paste into NEXUS tool Converts 2 vector of values for latitude and longitude into a format you can paste into NEXUS tool lat/lon site selection box

Usage

```
latlon2nexus(lat, lon)
```

Arguments

`lat`	vector of latitudes
`lon`	vector of longitudes

Value

a single character string that has all the csv pairs, with a semicolon between each pair and the next like 30.01,-90.61; 30.26,-90.95; 30.51,-91.23

Examples

```
lat_example = c(30.01,30.26,30.51)
lon_example = c(-90.61,-90.95,-91.23)
latlon2nexus(lat_example,lon_example)
```

`latloncsv2nexus` *helper function - combine lat/lon values to paste into NEXUS tool Converts vector of comma-separated values for latitude and longitude into a format you can paste into NEXUS tool lat/lon site selection box*

Description

helper function - combine lat/lon values to paste into NEXUS tool Converts vector of comma-separated values for latitude and longitude into a format you can paste into NEXUS tool lat/lon site selection box

Usage

```
latloncsv2nexus(latloncsv)
```

Arguments

`latloncsv`	a vector of comma-separated values with lat,lon

Value

a single character string that has all the csv pairs, with a semicolon between each pair and the next

Examples

```
latloncsv_example = c("30.01,-90.61", "30.26,-90.95", "30.51,-91.23")
latloncsv2nexus(latloncsv_example)
```

latlon_as.numeric *Strip non-numeric characters from a vector*

Description

Remove all characters other than minus signs, decimal points, and numeric digits

Usage

```
latlon_as.numeric(x)
```

Arguments

x	vector of something that is supposed to be numbers like latitude or longitude and may be a character vector because there were some other characters like tab or space or percent sign or dollar sign

Details

Useful if latitude or longitude vector has spaces, tabs, etc. CAUTION - Assumes stripping those out and making it numeric will fix whatever problem there was and end result is a valid set of numbers. Inf etc. are turned into NA values. Empty zero length string is turned into NA without warning. NA is left as NA. If anything other than empty or NA could not be interpreted as a number, it returns NA for those and offers a warning.

Value

numeric vector same length as x

See Also

latlon_df_clean() latlon_infer() latlon_is.valid() latlon_as.numeric()

Examples

```
latlon_as.numeric(c("-97.179167000000007", " -94.0533", "-95.152083000000005"))
latlon_as.numeric(-3:3)
latlon_as.numeric(c(1:3, NA))
latlon_as.numeric(c(1, 'asdf'))
latlon_as.numeric(c(1, ''))
latlon_as.numeric(c(1, '', NA))
latlon_as.numeric(c('aword', '$b'))
latlon_as.numeric(c('-10.5%', '<5', '$100'))
latlon_as.numeric(c(Inf, 1))
```

`latlon_df_clean` *Find and clean up latitude and longitude columns in a data.frame*

Description

Utility to identify lat and lon columns, renaming and cleaning them up.

Usage

```
latlon_df_clean(df)
```

Arguments

`df`	data.frame With columns lat and lon or names that can be interpreted as such - see latlon_infer()

Details

Tries to figure out which columns seem to have lat lon values, renames those in the data.frame. Cleans up lat and lon values (removes extra characters, makes numeric)

Value

Returns the same data.frame but with relevant colnames changed to lat and lon, and invalid lat or lon values cleaned up if possible or else replaced with NA

See Also

latlon_df_clean() latlon_infer() latlon_is.valid() latlon_as.numeric()

Examples

```
#  x <- latlon_df_clean(x)
```

`latlon_infer` *guess which columns have lat and lon based on aliases like latitude, FacLat, etc.*

Description

guess which columns have lat and lon based on aliases like latitude, FacLat, etc.

Usage

```
latlon_infer(mycolnames)
```

Arguments

`mycolnames`	e.g., colnames(x) where x is a data.frame from read.csv

Value

returns all of mycolnames except replacing the best candidates with lat and lon

See Also

latlon_df_clean() latlon_infer() latlon_is.valid() latlon_as.numeric()

Examples

```
latlon_infer(c('trilat', 'belong', 'belong')) # warns if no alias found,
  #  but doesnt warn of dupes in other terms, just preferred term.
latlon_infer(c('a', 'LONG', 'Longitude', 'lat')) # only the best alias is converted/used
latlon_infer(c('a', 'LONGITUDE', 'Long', 'Lat')) # only the best alias is converted/used
latlon_infer(c('a', 'longing', 'Lat', 'lat', 'LAT')) # case variants of preferred are
    # left alone only if lowercase one is found
latlon_infer(c('LONG', 'long', 'lat')) # case variants of a single alias are
    # converted to preferred word (if pref not found), creating dupes!  warn!
latlon_infer(c('LONG', 'LONG')) # dupes of an alias are renamed and still are dupes! warn!
latlon_infer(c('lat', 'lat', 'Lon')) # dupes left as dupes but warn!
```

`latlon_is.valid` *Validate latitudes and longitudes*

Description

Check each latitude and longitude value to see if they are NA or outside expected numeric ranges (based on approx ranges of lat lon seen among block internal points dataset) lat must be between 17.5 and 71.5, and lon must be (between -180 and -65) OR (between 172 and 180)

Usage

```
latlon_is.valid(lat, lon)
```

Arguments

`lat`	vector of latitudes in decimal degrees
`lon`	numeric vector of longitudes in decimal degrees, same length

Value

logical vector, one element per lat lon pair (location)

See Also

`latlon_df_clean()` `latlon_infer()` `latlon_is.valid()` `latlon_as.numeric()`

Examples

```
 ## Not run:
 table(latlon_is.valid(
   lat = EJAMblockdata::blockpoints$lat,
   lon = EJAMblockdata::blockpoints$lon)
   )
  ##      TRUE
  ## 8,174,955

## End(Not run)
```

`linkify` *make html link that opens in new tab*

Description

make html link that opens in new tab

Usage

```
linkify(url, text)
```

Arguments

`url`	string that is URL
`text`	string that is label

Value

linkify('epa.gov','EPA') returns `"<a href=\"epa.gov\", target=\"_blank\">EPA</a>"`

`locate_by_id` *query FRS API to find EPA facilities by registry ID or program ID*

Description

Uses the Facility Registry System (FRS) API to find sites by ID. This uses an API to find sites, but it is faster to look in a table if that FRS dataset is already loaded in an app, for example.

Usage

```
locate_by_id(id, type = "frs", ...)
```

Arguments

`id`	vector of one or more character strings that must be registry IDs (default) or program IDs
`type`	either frs (default) or program. frs means all are registry_id and program means all are pgm_sys_id
`...`	passed through to `locate_by_id1()`

Value

data.frame with one row per queried id, columns as returned by API but lat lon instead of Latitude83 Longitude83

See Also

`locate_by_id1()` and alias is `get_facility_info_via_FRS()`

Examples

```
 ## Not run:

 ids <- EJAMejscreenapi::testids_program_sys_id
 # ids <- c('ILR000128264','600039382','TXR1592DZ','TSCA8851',
 #    'CT0000000900908716', 'CEDRI10043548',  'CO0000000812305826')
 # ids <- c('ILR000128264','600039382')
 locate_by_id(ids, type = 'program')   # stopped working xxx ***********

sites_found_by_registry_id <- locate_by_id(testids_registry_id[1],
  type='frs')
sites_found_by_program_id  <- locate_by_id(testids_program_sys_id[1],
  type='program')
data.frame(example_REGID  <- t(sites_found_by_registry_id))
data.frame(example_PGMID  <- t(sites_found_by_program_id))
# Finding several facilities is slow:
sites_found_by_registry_id <- locate_by_id(testids_registry_id,
  type='frs')
sites_found_by_program_id  <- locate_by_id(testids_program_sys_id,
  type='program')
names(sites_found_by_program_id)
sites_found_by_registry_id[,c('RegistryId', 'lon','lat')]
## and  just to show agreement:
cbind(
 testids_program_sys_id,
 frs = EJAMfrsdata::frs[match(
   sites_found_by_program_id$RegistryId,
   EJAMfrsdata::frs$REGISTRY_ID
   ),
   c('PGM_SYS_ACRNMS',  'REGISTRY_ID')],
 api = sites_found_by_program_id[, c('RegistryId', 'lat', 'lon')]
)

# Leaflet map of sites, with a popup when one clicks on a site
df <- sites_found_by_registry_id
others <- EJAMfrsdata::frs[sample(1:NROW(EJAMfrsdata::frs), 5000),
 c('lon', 'lat')]
  leaflet::leaflet(df[4:6,]) |> leaflet::addTiles() |>
    leaflet::addMarkers(popup = popup_from_df(df[4:6,])) |>
    leaflet::addCircles(lng = others$lon, lat=others$lat,
      radius = 3*meters_per_mile, color = 'gray')

# Map with clickable icons
leaflet::leaflet(df) |> leaflet::addTiles() |>
# leaflet::addCircleMarkers(lng=~lon, lat=~lat, radius = 3) |>
 leaflet::addMarkers(popup = popup_from_df(df),
 icon = icons(iconUrl = './www/factory1.svg',
  iconWidth = 30, iconAnchorX = 14, iconAnchorY = 18)
)
 # Simple map of sites queried (among a sample of US EPA-regulated sites):

xl <- c(-125,-66); yl <- c(17,50) # just continental US plus PR
plot(EJAMfrsdata::frs[sample(1:NROW(EJAMfrsdata::frs),5000),c('lon','lat')],
 xlim=xl, ylim=yl, col='gray')
graphics::points(cbind(
 LONG=sites_found_by_program_id$lon,
 LAT=sites_found_by_program_id$lat),
 col='red', pch=16)

## End(Not run)
```

`locate_by_id1` *Helper function to query FRS API to find 1 EPA facility*

Description

Uses the Facility Registry System (FRS) API to find a site by registry ID or program ID. This uses an API to find sites, but it is faster to look in a table if that FRS dataset is already loaded in an app, for example.

Usage

```
locate_by_id1(id, type = "frs", ...)
```

Arguments

`id`	one character string that must be a registry ID (default) or program ID
`type`	either frs (default) which means registry_id or program which means pgm_sys_id
`...`	passed through to url_by_id()

Value

a 1 row data.frame, columns as returned by the API, but lat lon instead of Latitude83 Longitude83 ("RegistryId", "FacilityName", "LocationAddress", "CityName", "CountyName", "StateAbbr", "ZipCode", "FIPSCode", "lat", "lon")

See Also

locate_by_id() and url_by_id()

`lonlat_any_format` *Get latitude and longitude from a file, a data.frame, or vectors of lon,lat See notes on `read_and_clean_points()`*

Description

Get latitude and longitude from a file, a data.frame, or vectors of lon,lat See notes on `read_and_clean_points()`

Usage

```
lonlat_any_format(x, y)
```

Arguments

`x`	A filename (csv or xlsx, with path), or data.frame, or vector of longitudes. File or data.frame must have columns called lon and lat, or something that can be inferred to be that by latlon_infer()
`y`	If x is a vector of longitudes, y must be the latitudes. Ignored otherwise.

Value

A data.frame that has at least columns lon and lat (and others if they were in x)

See Also

latlon_df_clean()

Examples

```
lonlat_any_format(system.file("testdata/Sample12.xlsx", package="EJAMejscreenapi"))
lonlat_any_format(system.file("testdata/testpoints_05.csv", package="EJAMejscreenapi"))
lonlat_any_format(testpoints_50[1:6,] )
lonlat_any_format(testpoints_50[1:6, c('lat','lon')] )
lonlat_any_format(x=testpoints_50$lon[1:6], y=testpoints_50$lat[1:6] )
```

`make.popups.api` *Create the popup text for maps of EJ results*

Description

This creates the HTML text that appears in popup windows when you click on a site on the map, when viewing the results of EJ analysis of each site. THIS IS CURRENTLY HARD CODED TO USE EJScreen VARIABLE NAMES.

Usage

```
make.popups.api(
  out,
  linkcolname = "EJScreenPDF",
  linkcolname2 = "EJScreenMAP",
  verbose = FALSE
)
```

Arguments

`out`	raw data in data.frame form, with results of EJ buffer analysis
`linkcolname`	Name of one column in the table that has links to some URL
`linkcolname2`	Another like linkcolname
`verbose`	TRUE or FALSE, can see more details reported when function is used.

Value

HTML ready to be used for map popups

Examples

```
## Not run:
  out <- testoutput_ejscreenapi_plus_50
  x <- make.popups.api(out)
  popup_print(x)

## End(Not run)
```

`makenumericdfFORSHINY` *convert character columns back to numeric if they were meant to be numbers*

Description

removes some things like percent sign and less than sign and N/A and the word miles too

Usage

```
makenumericdfFORSHINY(x)
```

Arguments

`x`	data.frame from ejscreen api output

Value

data.frame

`mapfast` *quick simple leaflet map of data.frame with lat lon*

Description

quick simple leaflet map of data.frame with lat lon

Usage

```
mapfast(mydf, radius = 3, column_names = "all", labels = column_names)
```

Arguments

`mydf`	data.frame or data.table with lat and lon columns or columns that `latlon_infer()` can infer to be that
`radius`	in miles, converted to meters and passed to leaflet::addCircles()
`column_names`	If "ej" then nice popup made based on just key EJScreen indicators. If "all" then every column in the entire mydf table is shown in the popup. If a vector of colnames, only those are shown in popups.
`labels`	The labels used before the column_names, for map popups, like label: column_name (ignored if column_names is ej or all)

Value

plots a leaflet map with popups with all the columns from mydf

See Also

`popup_from_df()` `mapfastej()` `mapfast()`

Examples

```
  ## Not run:
 mapfast(testpoints_1000)

 mydf <- EJAMfrsdata::frs[sample(1:NROW(EJAMfrsdata::frs), 1000), 1:5]
 mapfast(mydf)

 mapfastej(testoutput_ejscreenapi_plus_50)
 mapfast(testoutput_ejscreenapi_plus_50, column_names = 'ej')
 mapfast(testoutput_ejscreenapi_plus_50)

## End(Not run)
```

`map_headernames` *CRITICAL DATASET WITH METADATA ABOUT ALL VARIABLES/ INDICATORS*

Description

CRITICAL DATASET WITH METADATA ABOUT ALL VARIABLES/ INDICATORS

Details

THIS IMPORTANT TABLE STORES INFORMATION ABOUT THE NAMES OF VARIABLES, INCLUDING ALTERNATIVE VERSIONS OF THE NAMES AS USED IN GEODATABASE FILES, IN THE CODE, SHORT VERSIONS FOR LABELS OF GRAPHICS, LONG VERSIONS TO PROVIDE FULL DESCRIPTIONS OF THE VARIABLES, TYPE OF VARIABLE FOR PURPOSES OF GROUPING SIMILAR ONES, ETC ETC

`meters_per_mile` *how many meters are in one mile (for conversions between units)*

Description

how many meters are in one mile (for conversions between units)

`near_eachother` *which points are near any of the others in list?*

Description

which points are near any of the others in list?

Usage

```
near_eachother(lon, lat, distance, or_tied = FALSE)
```

Arguments

`lon`	longitude
`lat`	latitude
`distance`	distance between points in miles to check
`or_tied`	if TRUE, checks if less than or equal to distance, otherwise if less than

popup_from_df — *Simple map popup from a data.frame, one point per row Creates popup that leaflet::addPopups can use.*

Description

Simple map popup from a data.frame, one point per row Creates popup that leaflet::addPopups can use.

Usage

```
popup_from_df(x, column_names = names(x), labels = column_names, n = "all")
```

Arguments

x	data.frame with info to be shown in map popups
column_names	default is all, or a vector of column names from x to use
labels	default is colnames(x) - vector used to label the elements in the popup. Must be same length as column_names
n	Show the first n columns of mypoints, in popup. "all" means all of them.

Details

Each popup is made from one row of the data.frame. Each popup has one row of text per column of the data.frame

Value

A vector of strings, one per row or map point, with a line break separating column elements

Examples

```
 df <- structure(list(
   RegistryId = c("110071102551", "110015787683"),
   FacilityName = c("USDOI FWS AK MARITIME NWR etc", "ADAK POWER PLANT"),
   LocationAddress=c("65 MI W. OF ADAK NAVAL FACILITY","100 HILLSIDE BLVD"),
   CityName = c("ADAK", "ADAK"),
   CountyName = c("ALEUTIAN ISLANDS", "ALEUTIANS WEST"),
   StateAbbr = c("AK", "AK"),
   ZipCode = c("99546", "99546"),
   FIPSCode = c("02010", "02016"),
   lat = c(51.671389,51.8703), lon = c(-178.051111, -176.659),
   SupplementalLocation = c(NA_character_,NA_character_)),
   row.names = 1:2, class = "data.frame")
 leaflet::leaflet(df) |> leaflet::addTiles() |>
   leaflet::addPopups(popup = popup_from_df(df))
```

`popup_from_uploadedpoints` *make simple popups for map to show info about uploaded points*

Description

make simple popups for map to show info about uploaded points

Usage

```
popup_from_uploadedpoints(mypoints, n = "all")
```

Arguments

`mypoints`	data.frame (or tibble?) with lat and lon columns preferably
`n`	Show the first n columns of mypoints, in popup. "all" means all of them.

Value

popups vector to be used in leaflet maps

See Also

`popup_from_df()`

`popup_print` *Helper function to view popup info in an interactive session - easier format to view*

Description

Helper function to view popup info in an interactive session - easier format to view

Usage

```
popup_print(x, linkregex = "<a href.*>(.*)<.*", linksimple = "\\1")
```

Arguments

`x`	output of `make.popups.api()`
`linkregex`	see source
`linksimple`	see source

See Also

`make.popups.api()`

`prep_for_excel` *Format batch results for excel See EJAM code related to this also!*

Description

Format batch results for excel See EJAM code related to this also!

Usage

```
prep_for_excel(
  df,
  hyperlink_cols = NULL,
  heatmap_colnames = NULL,
  heatmap_cuts = c(80, 90, 95),
  heatmap_colors = c("yellow", "orange", "red")
)
```

Arguments

`df`	data.frame, table of batch buffer results
`hyperlink_cols`	vector of names of columns in df to get treated as hyperlinks in excel
`heatmap_colnames`	vector of names of columns in df to apply conditional formatting to, by coloring like a heatmap.
`heatmap_cuts`	vector of color names for heatmap bins, same length as heatmap_cuts, where first color is for those >= 1st cutpoint, but <2d, second color is for those >=2d cutpoint but <3d, etc.
`heatmap_colors`	vector of colors corresponding to cuts

Value

A workbook via openxlsx::writeData() ready to be saved via openxlsx::saveWorkbook()

`ratios_to_avg` *get ratios of each site's scores to US means (using output of batch buffering)*

Description

get ratios of each site's scores to US means (using output of batch buffering)

Usage

```
ratios_to_avg(
  out,
 evarnames = c("pm", "o3", "cancer", "resp", "dpm", "pctpre1960", "traffic.score",
   "proximity.npl", "proximity.rmp", "proximity.tsdf", "proximity.npdes", "ust"),
  dvarnames = c("Demog.Index", "pctmin", "pctlowinc", "pctlths", "pctlingiso",
    "pctunder5", "pctover64", "pctunemployed"),
  zone.prefix = "us",
  avg.evarnames = paste0(zone.prefix, ".avg.", evarnames),
  avg.dvarnames = paste0(zone.prefix, ".avg.", dvarnames)
)
```

Arguments

`out`	data.frame output from `ejscreenapi_plus()` or from `ejscreeapi()` or `doaggregate()`, one row per buffer or site, and columns for indicators named in evarnames, dvarnames, avg.evarnames, avg.dvarnames
`evarnames`	vector of variable names for environmental indicators in out, like pm, o3, etc.
`dvarnames`	vector of variable names for demographic indicators in out, such as pctlowinc
`zone.prefix`	us or state, must fit with colnames in out such as us.avg.pctlowinc
`avg.evarnames`	use only if user-specific variable names are in out, with defaults like us.avg.pm
`avg.dvarnames`	use only if user-specific variable names are in out, with defaults like us.avg.pctlowinc

Details

Should recode to use variable name defaults from package not hardcoded here.

Value

a list with ratios_d and ratios_e which are vectors of numbers the lengths of dvarnames and evarnames

Examples

```
  pts <- data.frame(
   siteid = 1:2,
   sitename = c("example site A", "example site B"),
   lon = c(-91.132107, -91.09),
   lat = c(30.494982,   30.45)
  )
  ## Not run:
   out <- (ejscreenapi_script(pts=pts, radius = 1,
    save_table = FALSE, save_map = FALSE, save_plot = FALSE))$table
 out <- ejscreenapi_plus(pts,radius = 1, mapping_for_names=map_headernames)
   boxplots_ratios(ratios_to_avg(out)$ratios_d,"pctlowinc","% low income",
     wheretext="Within a mile of")

## End(Not run)
```

`read_and_clean_points` *Read .csv or .xlsx of lat/lon points or facility IDs*

Description

Read .csv or .xlsx of lat/lon points or facility IDs

Usage

```
read_and_clean_points(filepath, default_points = NULL)
```

Arguments

`filepath`	filename that can include path
`default_points`	what to return if no matches

Details

*** THIS WOULD REPLACE SOME OF THE CODE IN server.R or maybe rename this to latlon_or_naics_or_id_read()?? and merge with `latlon_any_format()` so one can read in a file or table or vectors - any format - and the other figures out if it is latlon or program IDs or registry IDs (but not NAICS?) and I guess turns those into lat, lon, siteid as below.

Value

data.frame with lat, lon, etc. columns

See Also

`read_csv_or_xl()`

`read_csv_or_xl` *Read table of data from .csv or .xlsx Excel file Read simple table from csv or xls or xlsx. For excel format, must be simple table on first tab, one row for header (column names), data itself starting in first cell of second row, like A2, and all other rows and columns must be empty.*

Description

Read table of data from .csv or .xlsx Excel file Read simple table from csv or xls or xlsx. For excel format, must be simple table on first tab, one row for header (column names), data itself starting in first cell of second row, like A2, and all other rows and columns must be empty.

Usage

```
read_csv_or_xl(fname, show_col_types = FALSE)
```

Arguments

`fname`	full path to folder and filename
`show_col_types`	FALSE makes it print less to console as it reads using readr::read_csv()

Value

data.frame with contents of table it read

`run_app` *Run the Shiny Application*

Description

Allows package to be a Shiny app and package at the same time.

Usage

```
run_app(
  onStart = NULL,
  options = list(),
  enableBookmarking = "server",
  uiPattern = "/",
  ...
)
```

Arguments

`onStart`	A function that will be called before the app is actually run. This is only needed for `shinyAppObj`, since in the `shinyAppDir` case, a `global.R` file can be used for this purpose.
`options`	Named options that should be passed to the `runApp` call (these can be any of the following: "port", "launch.browser", "host", "quiet", "display.mode" and "test.mode"). You can also specify `width` and `height` parameters which provide a hint to the embedding environment about the ideal height/width for the app.
`enableBookmarking`	Can be one of `"url"`, `"server"`, or `"disable"`. The default value, `NULL`, will respect the setting from any previous calls to `enableBookmarking()`. See `enableBookmarking()` for more information on bookmarking your app.
`uiPattern`	A regular expression that will be applied to each `GET` request to determine whether the `ui` should be used to handle the request. Note that the entire request path must match the regular expression in order for the match to be considered successful.
`...`	arguments to pass to golem_opts. See `?golem::get_golem_options` for more details.

Details

Normally R Shiny apps are not R packages - The server just sources all .R files found in the /R/ folder, and then runs what is found in app.R (if that is found / it is a one-file Shiny app). This R Shiny app, however, is shared as an R package, via the golem package approach, which provides the useful features of a package and useful features that the golem package enables.

There is still an app.R script in the package root – note there is no function called app() – which lets RStudio Connect source the app.R script to launch this shiny app.

The way this works is that there is a file called

_disable_autoload.R in the /R/ folder

to tell the server to not source all the source .R files, since they are already in the installed package. Then they get loaded from the package because the app.R script here says this:

```
pkgload::load_all(export_all = FALSE,helpers = FALSE,attach_testthat = FALSE)
```

with the shinyApp() call wrapped in shiny::runApp() rather than in app()

Also, app_runYYYY() is the same as YYYY::run_app() in case that is useful.

See https://thinkr-open.github.io/golem/

---

speedmessage *estimate how long it will take to get buffer batch results*

Description

estimate how long it will take to get buffer batch results

Usage

```
speedmessage(n, perhourslow = 1000, perhourfast = 12000, perhourguess = 6000)
```

Arguments

`n`	number of points to buffer at
`perhourslow`	n per hour if slow (conservative estimate of time needed)
`perhourfast`	n per hour if fast
`perhourguess`	n per hour best guess

See Also

`speedreport()`

---

speedreport *helper function that reports on how long buffering took*

Description

helper function that reports on how long buffering took

Usage

```
speedreport(start, end, n)
```

Arguments

`start`	start time
`end`	end time
`n`	how many buffers were completed

Value

text string summarizing the speed

See Also

`speedmessage()`

`testids_program_sys_id` *test data, string vector of EPA FRS Program System ID numbers*

Description

test data, string vector of EPA FRS Program System ID numbers

Details

"7-0540-00003", "354362", "1513529", "485659", "LAG750956", "CAC002995519", "3601252181", "3601439158"

`testids_registry_id` *test data, vector of EPA FRS Registry ID numbers*

Description

test data, vector of EPA FRS Registry ID numbers

Details

110071293460, 110070874073, 110070538057, 110044340807, 110030509215, 110019033810, 110056111559, 110056982323

`testoutput_ejscreenapi_plus_50` *test data examples of output from* `ejscreenapi_plus()`

Description

test data examples of output from `ejscreenapi_plus()`

Details

Just for convenience, installed with the package. Has header row plus 50 rows, and about 200+ columns of buffer summary results.

testpoints_50 *test data examples of what could be input to functions that needs points specified by lat lon*

Description

test data examples of what could be input to functions that needs points specified by lat lon

Details

Just for convenience, these are installed with the package, but are the equivalent of results of read.csv() for each test data file.

url_by_id *Get the URLs to use to query FRS API to find EPA facilities by ID*

Description

This uses an API to find sites, but it is faster to look in a table if that FRS dataset is already loaded in an app, for example.

Usage

```
url_by_id(id, type = "frs", ...)
```

Arguments

`id`	vector of one or more character strings with pgm_sys_id or registry_id values (all need to be the same type, as defined by type parameter). Program ids are like "VA0088986" and frs ids are like "110015787683"
`type`	one word, applies to all. default is frs but can be program or the word other.
`...`	appended to the end of the URL as-is, useful if type is other, for example

Details

For details on FRS API, see https://www.epa.gov/frs/frs-rest-services and examples at https://www.epa.gov/frs/frs-rest-services#ex1 and more at https://www.epa.gov/frs/frs-rest-services#appendixa For example:
https://frs-public.epa.gov/ords/frs_public2/frs_rest_services.get_facilities?pgm_sys_id=VA0088986
https://frs-public.epa.gov/ords/frs_public2/frs_rest_services.get_facilities?registry_id=110010912496
Note: API URL for internal use at EPA appears to be different than public one?

Value

vector of URLs as strings, same length as id parameter

See Also

locate_by_id()

Examples

```
 ## Not run:
    url_by_id(testids_registry_id)
    browseURL(url_by_id(testids_registry_id)[1])

## End(Not run)
```

`varname2color` *helper function - for color coding excel sheet columns*

Description

helper function - for color coding excel sheet columns

Usage

```
varname2color(varname, varnameinfo)
```

Arguments

`varname`	things like us.avg.pctlowinc
`varnameinfo`	data.frame with info on type of each variable

Value

vector of colors

See Also

`varname2vartype()` `vartype2color()` `varname2color()`

`varname2vartype` *helper function - given indicator names, look up what type each is*

Description

helper function - given indicator names, look up what type each is

Usage

```
varname2vartype(varname, varnameinfo)
```

Arguments

`varname`	vector of 1 or more names
`varnameinfo`	data.frame with info on type of each variable

Details

The types are things like raw data count for indicator, average, percentile, etc. Variable names are stored in column of varnameinfo called newnames_ejscreenapi Types are stored in column of varnameinfo called jsondoc_vartype

Value

vector same size as varname

See Also

`varname2vartype()` `vartype2color()` `varname2color()`

`varsinfo_ECHO_API` *Table of metadata about the variables available via the ECHO API*

Description

Table of metadata about the variables available via the ECHO API

Details

The ECHO API provides access to facilities in the EPA Facility Registry Services (FRS). Metadata were obtained from here: https://echodata.epa.gov/echo/echo_rest_services.metadata?output=JSON This table has some useful information about selected variables that are available. It notes which are the most useful for EJAM-related work, and notes the name and ID of the variable, which is needed to request that info via the API.

Also see `get_facility_info_via_ECHO()` This table has 316 rows and these columns: "ObjectName", "desc", "ColumnID", "critical", "best", "useful", "programid", "ej", "ColumnName", "DataType", "DataLength", "Description", "all" For example one row has this information: ObjectName "SDWAIDs" desc "A unique 9-character ID assigned for each public w" ColumnID "24" Critical "FALSE" best "TRUE" useful "TRUE" programid "TRUE" ej "FALSE" ColumnName "SDWA_IDS" etc.

To see a full list of variables of interest:

subset(EJAMejscreenapi::varsinfo_ECHO_API, EJAMejscreenapi::varsinfo_ECHO_API$useful)

`vartype2color` *helper function - assign fill color to shade excel cells by indicator type Use color shading to make spreadsheet easier to use, grouping the indicators*

Description

helper function - assign fill color to shade excel cells by indicator type Use color shading to make spreadsheet easier to use, grouping the indicators

Usage

```
vartype2color(vartype)
```

Arguments

`vartype`	must be one found in varnameinfo$jsondoc_vartype, ie "percentile", "average", or "raw data for indicator" NA if not found.

Value

vector of colors like c('lightorange', 'gray')

See Also

`varname2vartype()` `vartype2color()` `varname2color()`

Index