Package 'soilDB'

October 21, 2020

Type Package

Title Soil Database Interface

Version 2.5.8

Date 2020-10-20

Author Dylan Beaudette [aut],
Jay Skovlin [aut],
Stephen Roecker [aut],
Andrew Brown [aut, cre]

Maintainer Andrew Brown <andrew.g.brown@usda.gov>

Description A collection of functions for reading data from USDA-NCSS soil databases.

License GPL (>= 3)

LazyLoad yes

Depends R (>= 3.5.0)

Imports aqp, grDevices, graphics, stats, utils, plyr, xml2, sp, reshape2, raster, curl, lattice, methods, data.table

Suggests rgdal, jsonlite, RODBC, httr, sf, rgeos, rvest, testthat, stringr, latticeExtra, RCurl, XML, ggplot2, gridExtra, viridis, mapview, rasterVis

Repository CRAN

URL http://ncss-tech.github.io/AQP/

BugReports https://github.com/ncss-tech/soilDB/issues

RoxygenNote 7.1.1

NeedsCompilation no

Date/Publication 2020-10-21 05:40:02 UTC

R topics documented:

`soilDB-package` *Soil Database Interface*

Description

This package provides methods for extracting soils information from local PedonPC and AK Site databases (MS Access format), local NASIS databases (MS SQL Server), and the SDA webservice. Currently USDA-NCSS data sources are supported, however, there are plans to develop interfaces to outside systems such as the Global Soil Mapping project.

Details

It can be difficult to locate all of the dependencies required for sending/processing SOAP requests, especially on UNIX-like operating systems. Windows binary packages for the dependencies can be found here. See `fetchPedonPC` for a simple wrapper function that should suffice for typical site/pedon/hz queries. An introduction to the soilDB package can be found here.

Author(s)

J.M. Skovlin and D.E. Beaudette

See Also

`fetchPedonPC`, `fetchNASIS`, `SDA_query`, `loafercreek`

`estimateColorMixture` *Estimate color mixtures using weighted average of CIELAB color coordinates*

Description

Estimate color mixtures using weighted average of CIELAB color coordinates

Usage

```
estimateColorMixture(x, wt = "pct", backTransform = FALSE)
```

Arguments

`x`	data.frame, typically from NASIS containing at least CIE LAB ('L', 'A', 'B') and some kind of weight
`wt`	fractional weights, usually area of hz face
`backTransform`	logical, should the mixed sRGB representation of soil color be transformed to closest Munsell chips? This is performed by aqp::rgb2Munsell default: `FALSE`

Value

A data.frame containing estimated color mixture

Note

See `mixMunsell` for a more realistic (but slower) simulation of subtractive mixing of pigments.

Author(s)

D.E. Beaudette

`estimateSTR` *Estimate Soil Temperature Regime*

Description

Estimate soil temperature regime (STR) based on mean annual soil temperature (MAST), mean summer temperature (MSST), mean winter soil temperature (MWST), presence of O horizons, saturated conditions, and presence of permafrost. Several assumptions are made when O horizon or saturation are undefined.

Usage

```
estimateSTR(mast, mean.summer, mean.winter, O.hz = NA, saturated = NA, permafrost = FALSE)
```

Arguments

mast	vector of mean annual soil temperature (deg C)
mean.summer	vector of mean summer soil temperature (deg C)
mean.winter	vector of mean winter soil temperature (deg C)
O.hz	logical vector of O horizon presence / absence
saturated	logical vector of seasonal saturation
permafrost	logical vector of permafrost presence / absence

Details

Related tutorial.

Value

Vector of soil temperature regimes.

Author(s)

D.E. Beaudette

References

Soil Survey Staff. 2015. Illustrated guide to soil taxonomy. U.S. Department of Agriculture, Natural Resources Conservation Service, National Soil Survey Center, Lincoln, Nebraska.

See Also

STRplot

Examples

```
# simple example
estimateSTR(mast=17, mean.summer = 22, mean.winter = 12)
```

---

fetchGDB *Load and Flatten Data from SSURGO file geodatabases*

Description

Functions to load and flatten commonly used tables and from SSURGO file geodatabases, and create soil profile collection objects (SPC).

Usage

```
fetchGDB(dsn = "gNATSGO_CONUS.gdb",
         WHERE = NULL,
         childs = TRUE,
         droplevels = TRUE,
         stringsAsFactors = TRUE
         )


get_legend_from_GDB(dsn = "gNATSGO_CONUS.gdb",
                    WHERE = NULL,
                    droplevels = TRUE,
                    stringsAsFactors = TRUE,
                    stats = FALSE
                    )

get_mapunit_from_GDB(dsn = "gNATSGO_CONUS.gdb",
                     WHERE = NULL,
                     droplevels = TRUE,
                     stringsAsFactors = TRUE,
                     stats = FALSE
                     )

get_component_from_GDB(dsn = "gNATSGO_CONUS.gdb",
                       WHERE = NULL,
                       childs = FALSE,
                       droplevels = TRUE,
                       stringsAsFactors = TRUE
                       )
```

Arguments

dsn	data source name (interpretation varies by driver - for some drivers, dsn is a file name, but may also be a folder, or contain the name and access credentials of a database); in case of GeoJSON, dsn may be the character string holding the geojson data. It can also be an open database connection.
WHERE	text string formatted as an SQL WHERE clause (default: FALSE)
childs	logical; if FALSE parent material and geomorphic child tables are not flattened and appended
droplevels	logical: indicating whether to drop unused levels in classifying factors. This is useful when a class has large number of unused classes, which can waste space in tables and figures.
stringsAsFactors	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

`stats` Return extended summary statistics (for legend or mapunit only)

Details

These functions return data from SSURGO file geodatabases with the use of a simple text string that formatted as an SQL WHERE clause (e.g. `WHERE = "areasymbol = 'IN001'"`. Any columns within the target table can be specified (except for fetchGDB() currently, which only targets the legend with the WHERE clause).

Value

A `data.frame` or `SoilProfileCollection` object.

Author(s)

Stephen Roecker

Examples

```
## replace `dsn` with path to your own geodatabase (SSURGO OR gNATSGO)
##
##
##  download CONUS gNATSGO from here:
##   https://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/survey/geo/?cid=nrcseprd1464625
##
##
# dsn <- "D:/geodata/soils/gNATSGO_CONUS.gdb"

# le <- get_legend_from_GDB(dsn = dsn, WHERE = "areasymbol LIKE '%'")

# mu <- get_mapunit_from_GDB(dsn = dsn, WHERE = "muname LIKE 'Miami%'")

# co <- get_component_from_GDB(dsn, WHERE = "compname = 'Miami'
#                              AND majcompflag = 'Yes'", childs = FALSE)

# f_in_GDB <- fetchGDB(WHERE = "areasymbol LIKE 'IN%'")
```

`fetchHenry` *Download Data from the Henry Mount Soil Temperature and Water Database*

Description

This function is a front-end to the REST query functionality of the Henry Mount Soil Temperature and Water Database.

Usage

```
fetchHenry(what='all', usersiteid = NULL, project = NULL, sso = NULL,
gran = "day", start.date = NULL, stop.date = NULL,
pad.missing.days = TRUE, soiltemp.summaries = TRUE)
```

Arguments

what	type of data to return: 'sensors': sensor metadata only | 'soiltemp': sensor metadata + soil temperature data | 'soilVWC': sensor metadata + soil moisture data | 'airtemp': sensor metadata + air temperature data | 'waterlevel': sensor metadata + water level data l'all': sensor metadata + all sensor data
usersiteid	(optional) filter results using a NASIS user site ID
project	(optional) filter results using a project ID
sso	(optional) filter results using a soil survey office code
gran	data granularity: "day", "week", "month", "year"; returned data are averages
start.date	(optional) starting date filter
stop.date	(optional) ending date filter
pad.missing.days	should missing data ("day" granularity) be filled with NA? see details
soiltemp.summaries	should soil temperature ("day" granularity only) be summarized? see details

Details

Filling missing days with NA is useful for computing and index of how complete the data are, and for estimating (mostly) unbiased MAST and seasonal mean soil temperatures. Summaries are computed by first averaging over Julian day, then averaging over all days of the year (MAST) or just those days that occur within "summer" or "winter". This approach makes it possible to estimate summaries in the presence of missing data. The quality of summaries should be weighted by the number of "functional years" (number of years with non-missing data after combining data by Julian day) and "complete years" (number of years of data with >= 365 days of non-missing data).

Value

a list containing:

sensors	a SpatialPointsDataFrame object containing site-level information
soiltemp	a data.frame object containing soil temperature timeseries data
soilVWC	a data.frame object containing soil moisture timeseries data
airtemp	a data.frame object containing air temperature timeseries data
waterlevel	a data.frame object containing water level timeseries data

Note

This function and the back-end database are very much a work in progress.

Author(s)

D.E. Beaudette

See Also

fetchSCAN

Examples

```
if(requireNamespace("curl") &
    curl::has_internet() &
    require(lattice)) {

  # get CA630 data as daily averages
  x <- fetchHenry(project='CA630', gran = 'day')

  # inspect data gaps
  levelplot(factor(!is.na(sensor_value)) ~ doy * factor(year) | name,
  data=x$soiltemp, col.regions=c('grey', 'RoyalBlue'), cuts=1,
  colorkey=FALSE, as.table=TRUE, scales=list(alternating=3),
  par.strip.text=list(cex=0.75), strip=strip.custom(bg='yellow'),
  xlab='Julian Day', ylab='Year')

}
```

---

fetchKSSL *Fetch KSSL Data*

Description

Download soil characterization and morphologic data via BBOX, MLRA, or soil series name query, from the KSSL database.

Usage

```
fetchKSSL(series=NA, bbox=NA, mlra=NA, pedlabsampnum=NA,
pedon_id=NA, pedon_key=NA, returnMorphologicData=FALSE, returnGeochemicalData=FALSE,
simplifyColors=FALSE, progress=TRUE)
```

Arguments

series	vector of soil series names, case insensitive
bbox	a single bounding box in WGS84 geographic coordinates e.g. c(-120,37,-122,38)
mlra	vector of MLRA IDs, e.g. "18" or "22A"
pedlabsampnum	vector of KSSL pedon lab sample number

pedon_id	vector of user pedon ID
pedon_key	vector of KSSL internal pedon ID
returnMorphologicData	logical, optionally request basic morphologic data, see details section
returnGeochemicalData	logical, optionally request geochemical, optical and XRD/thermal data, see details section
simplifyColors	logical, simplify colors (from morphologic data) and join with horizon data
progress	logical, optionally give progress when iterating over multiple requests

Details

This is an experimental interface to a subset for the most commonly used data from a snapshot of KSSL (lab characterization) and NASIS (morphologic) data.

Series-queries are case insensitive. Series name is based on the "correlated as" field (from KSSL snapshot) when present. The "sampled as" classification was promoted to "correlated as" if the "correlated as" classification was missing.

When `returnMorphologicData` is TRUE, the resulting object is a list. The standard output from `fetchKSSL` (`SoilProfileCollection` object) is stored in the named element "SPC". The additional elements are basic morphologic data: soil color, rock fragment volume, pores, structure, and redoximorphic features. There is a 1:many relationship between the horizon data in "SPC" and the additional dataframes in `morph`. See examples for ideas on how to "flatten" these tables.

When `returnGeochemicalData` is TRUE, the resulting object is a list. The standard output from `fetchKSSL` (`SoilProfileCollection` object) is stored in the named element "SPC". The additional elements are geochemical and mineralogy analysis tables, specifically: geochemical/elemental analyses "geochem", optical mineralogy "optical", and X-ray diffraction / thermal "xrd_thermal". `returnGeochemicalData` will include additional dataframes `geochem`, `optical`, and `xrd_thermal` in list result.

Setting `simplifyColors=TRUE` will automatically flatten the soil color data and join to horizon level attributes.

Function arguments (`series`, `mlra`, etc.) are fully vectorized except for `bbox`.

Value

a `SoilProfileCollection` object when `returnMorphologicData` is FALSE, otherwise a list.

Note

SoilWeb maintains a snapshot of these KSSL and NASIS data. The SoilWeb snapshot was developed using methods described here: https://github.com/dylanbeaudette/process-kssl-snapshot. Please use the link below for the live data.

Author(s)

D.E. Beaudette and A.G. Brown

References

http://ncsslabdatamart.sc.egov.usda.gov/

See Also

fetchOSD

Examples

```
if(requireNamespace("curl") &
    curl::has_internet()) {

    library(aqp)
    library(plyr)
    library(reshape2)

    # search by series name
    s <- fetchKSSL(series='auburn')

    # search by bounding-box
    # s <- fetchKSSL(bbox=c(-120, 37, -122, 38))

    # how many pedons
    length(s)

    # plot
    plotSPC(s, name='hzn_desgn', max.depth=150)

    ##
    ## morphologic data
    ##

    # get lab and morphologic data
    s <- fetchKSSL(series='auburn', returnMorphologicData = TRUE)

    # extract SPC
    pedons <- s$SPC

    ## automatically simplify color data
    s <- fetchKSSL(series='auburn', returnMorphologicData = TRUE, simplifyColors=TRUE)

    # check
    par(mar=c(0,0,0,0))
    plot(pedons, color='moist_soil_color', print.id=FALSE)

}
```

`fetchNASIS` *Fetch commonly used site/pedon/horizon or component data from NASIS.*

Description

Fetch commonly used site/pedon/horizon data or component from NASIS, returned as a SoilProfileCollection object.

Usage

```
fetchNASIS(from = 'pedons', url = NULL, SS=TRUE, rmHzErrors=TRUE, nullFragsAreZero=TRUE,
                  soilColorState='moist', lab=FALSE, fill = FALSE,
                  stringsAsFactors = default.stringsAsFactors()
                  )

getHzErrorsNASIS(strict=TRUE)
```

Arguments

`from`	determines what objects should fetched? ('pedons' | 'components' | 'pedon_report')
`url`	string specifying the url for the NASIS pedon_report (default: NULL)
`SS`	fetch data from the currently loaded selected set in NASIS or from the entire local database (default: TRUE)
`stringsAsFactors`	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have been set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)
`rmHzErrors`	should pedons with horizonation errors be removed from the results? (default: TRUE)
`nullFragsAreZero`	should fragment volumes of NULL be interpreted as 0? (default: TRUE), see details
`soilColorState`	which colors should be used to generate the convenience field 'soil_color'? ('moist' | 'dry')
`lab`	should the phlabresults child table be fetched with site/pedon/horizon data (default: FALSE)
`fill`	(fetchNASIS(from='components') only: include component records without horizon data in result? (default: FALSE)
`strict`	how strict should horizon boundaries be checked for consistency: TRUE=more | FALSE=less

Details

This function imports data from NASIS into R as a SoilProfileCollection object. It "flattens" NASIS pedon and component tables, including their child tables, into several more easily manageable data frames. Primarily these functions access the local NASIS database using an ODBC connection. However using the fetchNASIS() argument from = "pedon_report", data can be read from the NASIS Report 'fetchNASIS', as either a txt file or url. The primary purpose of fetchNASIS(from = "pedon_report") is to facilitate importing datasets larger than 8000+ pedons/components.

The value of `nullFragsAreZero` will have a significant impact on the rock fragment fractions returned by `fetchNASIS`. Set `nullFragsAreZero = FALSE` in those cases where there are many data-gaps and NULL rock fragment values should be interpreted as NULLs. Set `nullFragsAreZero = TRUE` in those cases where NULL rock fragment values should be interpreted as 0.

This function attempts to do most of the boilerplate work when extracting site/pedon/horizon or component data from a local NASIS database. Pedons that are missing horizon data, or have errors in their horizonation are excluded from the returned object, however, their IDs are printed on the console. Pedons with combination horizons (e.g. B/C) are erroneously marked as errors due to the way in which they are stored in NASIS as two overlapping horizon records.

See `getHzErrorsNASIS` for a simple approach to identifying pedons with problematic horizonation.

See the NASIS component tutorial, and NASIS pedon tutorial for more information.

Value

a SoilProfileCollection class object

Author(s)

D. E. Beaudette, J. M. Skovlin, and S.M. Roecker

`fetchNASISLabData` *Fetch lab data used site/horizon data from a PedonPC database.*

Description

Fetch KSSL laboratory pedon/horizon layer data from a local NASIS database, return as a SoilProfileCollection object.

Usage

```
fetchNASISLabData(SS = TRUE)
```

Arguments

`SS`	fetch data from the currently loaded selected set in NASIS or from the entire local database (default: TRUE)

Details

This function currently works only on Windows, and requires a 'nasis_local' ODBC connection.

Value

a SoilProfileCollection class object

Note

This function attempts to do most of the boilerplate work when extracting KSSL laboratory site/horizon data from a local NASIS database. Lab pedons that have errors in their horizonation are excluded from the returned object, however, their IDs are printed on the console. See `getHzErrorsNASIS` for a simple approach to identifying pedons with problematic horizonation.

Author(s)

J.M. Skovlin and D.E. Beaudette

See Also

`get_labpedon_data_from_NASIS_db`

---

fetchNASISWebReport *Extract component tables from a the NASIS Web Reports*

Description

Get, format, impute, and return component tables.

Usage

```
fetchNASISWebReport(projectname, rmHzErrors = FALSE, fill = FALSE,
                    stringsAsFactors = default.stringsAsFactors()
                    )
get_progress_from_NASISWebReport(mlrassoarea, fiscalyear, projecttypename)
get_project_from_NASISWebReport(mlrassoarea, fiscalyear)
get_project_correlation_from_NASISWebReport(mlrassoarea, fiscalyear, projectname)
get_projectmapunit_from_NASISWebReport(projectname,
                                  stringsAsFactors = default.stringsAsFactors()
                                       )
get_projectmapunit2_from_NASISWebReport(mlrassoarea, fiscalyear, projectname,
                                  stringsAsFactors = default.stringsAsFactors()
                                        )
get_legend_from_NASISWebReport(areasymbol,
                               droplevels = TRUE,
                               stringsAsFactors = default.stringsAsFactors()
                               )
get_mapunit_from_NASISWebReport(areasymbol,
                                droplevels = TRUE,
                                stringsAsFactors = default.stringsAsFactors()
                                )
```

```
get_component_from_NASISWebReport(projectname,
                                  stringsAsFactors = default.stringsAsFactors()
                                  )
get_chorizon_from_NASISWebReport(projectname, fill = FALSE,
                                 stringsAsFactors = default.stringsAsFactors()
                                 )
get_cosoilmoist_from_NASISWebReport(projectname, impute = TRUE,
                                  stringsAsFactors = default.stringsAsFactors()
                                    )
get_sitesoilmoist_from_NASISWebReport(usiteid)
```

Arguments

projectname	text string vector of project names to be inserted into a SQL WHERE clause (default: NA)
mlrassoarea	text string value identifying the mlra soil survey office areasymbol symbol inserted into a SQL WHERE clause (default: NA)
fiscalyear	text string value identifying the fiscal year inserted into a SQL WHERE clause (default: NA)
projecttypename	text string value identifying the project type name inserted into a SQL WHERE clause (default: NA)
areasymbol	text string value identifying the area symbol (e.g. "IN001" or "IN%") inserted into a SQL WHERE clause (default: NA)
usiteid	text string value identifying the user site id inserted into a SQL WHERE clause (default: NA)
impute	replace missing (i.e. NULL) values with "Not_Populated" for categorical data, or the "RV" for numeric data or 201 cm if the "RV" is also NULL (default: TRUE)
fill	should rows with missing component ids be removed NA (FALSE)
rmHzErrors	should pedons with horizonation errors be removed from the results? (default: FALSE)
stringsAsFactors	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have been set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)
droplevels	logical: indicating whether to drop unused levels in classifying factors. This is useful when a class has large number of unused classes, which can waste space in tables and figures.

Value

A data.frame or list with the results.

Author(s)

Stephen Roecker

Examples

```
if (requireNamespace("curl") &
    curl::has_internet() &
    require("aqp") &
    require("ggplot2") &
    require("gridExtra")
) {
  # query soil components by projectname
  test = fetchNASISWebReport(
    "EVAL - MLRA 111A - Ross silt loam, 0 to 2 percent slopes, frequently flooded"
  )
  test = test$spc

  # profile plot
  plot(test)

  # convert the data for depth plot
  clay_slice = horizons(slice(test, 0:200 ~ claytotal_l + claytotal_r + claytotal_h))
  names(clay_slice) <- gsub("claytotal_", "", names(clay_slice))

  om_slice = horizons(slice(test, 0:200 ~ om_l + om_r + om_h))
  names(om_slice) = gsub("om_", "", names(om_slice))

  test2 = rbind(data.frame(clay_slice, var = "clay"),
                data.frame(om_slice, var = "om")
  )

  h = merge(test2, site(test)[c("dmuiid", "coiid", "compname", "comppct_r")],
            by = "coiid",
            all.x = TRUE
  )

  # depth plot of clay content by soil component
  gg_comp <- function(x) {
    ggplot(x) +
      geom_line(aes(y = r, x = hzdept_r)) +
      geom_line(aes(y = r, x = hzdept_r)) +
      geom_ribbon(aes(ymin = l, ymax = h, x = hzdept_r), alpha = 0.2) +
      xlim(200, 0) +
      xlab("depth (cm)") +
      facet_grid(var ~ dmuiid + paste(compname, comppct_r)) +
      coord_flip()
  }
  g1 <- gg_comp(subset(h, var == "clay"))
  g2 <- gg_comp(subset(h, var == "om"))
```

```
  grid.arrange(g1, g2)


  # query cosoilmoist (e.g. water table data) by mukey
  # NA depths are interpreted as (???) with impute=TRUE argument
  x <- get_cosoilmoist_from_NASISWebReport(
    "EVAL - MLRA 111A - Ross silt loam, 0 to 2 percent slopes, frequently flooded"
  )

  ggplot(x, aes(x = as.integer(month), y = dept_r, lty = status)) +
    geom_rect(aes(xmin = as.integer(month), xmax = as.integer(month) + 1,
                  ymin = 0, ymax = max(x$depb_r),
                  fill = flodfreqcl)) +
    geom_line(cex = 1) +
    geom_point() +
    geom_ribbon(aes(ymin = dept_l, ymax = dept_h), alpha = 0.2) +
    ylim(max(x$depb_r), 0) +
    xlab("month") + ylab("depth (cm)") +
    scale_x_continuous(breaks = 1:12, labels = month.abb, name="Month") +
    facet_wrap(~ paste0(compname, ' (', comppct_r , ')')) +
    ggtitle(paste0(x$nationalmusym[1],
                   ': Water Table Levels from Component Soil Moisture Month Data'))


}
```

`fetchOSD` *Fetch Official Series Descriptions and summaries from SoilWeb API*

Description

This function fetches a variety of data associated with named soil series, extracted from the USDA-NRCS Official Series Description text files and detailed soil survey (SSURGO). These data are periodically updated and made available via SoilWeb.

Usage

```
fetchOSD(soils, colorState = "moist", extended = FALSE)
```

Arguments

`soils`	a character vector of named soil series; case-insensitive
`colorState`	color state for horizon soil color visualization: "moist" or "dry"
`extended`	if `TRUE` additional soil series summary data are returned, see details

Details

- overview of all soil series query functions
- competing soil series
- siblings

The standard set of "site" and "horizon" data are returned as a `SoilProfileCollection` object (`extended=FALSE`. The "extended" suite of summary data can be requested by setting `extended=TRUE`. The resulting object will be a `list` with the following elements:)

SPC `SoilProfileCollection` containing standards "site" and "horizon" data

competing competing soil series from the SC database snapshot

geomcomp empirical probabilities for geomorphic component, derived from the current SSURGO snapshot

hillpos empirical probabilities for hillslope position, derived from the current SSURGO snapshot

mtnpos empirical probabilities for mountain slope position, derived from the current SSURGO snapshot

terrace empirical probabilities for river terrace position, derived from the current SSURGO snapshot

flats empirical probabilities for flat landscapes, derived from the current SSURGO snapshot

pmkind empirical probabilities for parent material kind, derived from the current SSURGO snapshot

pmorigin empirical probabilities for parent material origin, derived from the current SSURGO snapshot

mlra empirical MLRA membership values, derived from the current SSURGO snapshot

climate experimental climate summaries from PRISM stack

metadata metadata associated with SoilWeb cached summaries

When using ‘extended=TRUE‘, there are a couple of scenarios in which series morphology contained in ‘SPC‘ do not fully match records in the associated series summaries (e.g. ‘competing‘).

1. **A query for soil series that exist entirely outside of CONUS (e.g. PALAU).** - Climate summaries are empty `data.frames` because these summaries are currently generated from PRISM. We are working on a solution.
2. **A query for data within CONUS, but OSD morphology missing due to parsing error (e.g. formatting, typos).** - Extended summaries are present but morphology missing from ‘SPC‘. A warning is issued.
3. **A query for multiple soil series, with one more more listed as "inactive" (e.g. BREADSPRINGS).** - Extended summaries are present but morphology missing from ‘SPC‘. A warning is issued.

These last two cases are problematic for analysis that makes use of morphology and extended data, such as outlined in this tutorial on competing soil series.

Value

a `SoilProfileCollection` object containing basic soil morphology and taxonomic information.

Author(s)

D.E. Beaudette

References

USDA-NRCS OSD search tools: https://www.nrcs.usda.gov/wps/portal/nrcs/detailfull/soils/home/?cid=nrcs142p2_053587

See Also

OSDquery, siblings

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

  # soils of interest
  s.list <- c('musick', 'cecil', 'drummer', 'amador', 'pentz',
              'reiff', 'san joaquin', 'montpellier', 'grangeville', 'pollasky', 'ramona')

  # fetch and convert data into an SPC
  s.moist <- fetchOSD(s.list, colorState='moist')
  s.dry <- fetchOSD(s.list, colorState='dry')

  # plot profiles
  # moist soil colors
  if(require("aqp")) {

    par(mar=c(0,0,0,0), mfrow=c(2,1))
    plot(s.moist, name='hzname',
         cex.names=0.85, axis.line.offset=-4)
    plot(s.dry, name='hzname',
         cex.names=0.85, axis.line.offset=-4)

    # extended mode: return a list with SPC + summary tables
    x <- fetchOSD(s.list, extended = TRUE, colorState = 'dry')

    par(mar=c(0,0,1,1))
    plot(x$SPC)
    str(x, 1)
  }
}
```

`fetchPedonPC` *Fetch commonly used site/horizon data from a PedonPC v.5 database.*

Description

Fetch commonly used site/horizon data from a version 5.x PedonPC database, return as a SoilProfileCollection object.

Usage

```
fetchPedonPC(dsn)
getHzErrorsPedonPC(dsn, strict=TRUE)
```

Arguments

`dsn`	The path to a PedonPC version 5.x database
`strict`	should horizonation by strictly enforced? (TRUE)

Details

This function currently works only on Windows.

Value

a SoilProfileCollection class object

Note

This function attempts to do most of the boilerplate work when extracting site/horizon data from a PedonPC or local NASIS database. Pedons that have errors in their horizonation are excluded from the returned object, however, their IDs are printed on the console. See `getHzErrorsPedonPC` for a simple approach to identifying pedons with problematic horizonation. Records from the 'taxhistory' table are selected based on 1) most recent record, or 2) record with the least amount of missing data.

Author(s)

D. E. Beaudette and J. M. Skovlin

See Also

`get_hz_data_from_pedon_db`

fetchRaCA *Get Rapid Carbon Assessment (RaCA) data*

Description

Get Rapid Carbon Assessment (RaCA) data via state, geographic bounding-box, RaCA site ID, or series query from the SoilWeb API.

Usage

```
fetchRaCA(
  series = NULL,
  bbox = NULL,
  state = NULL,
  rcasiteid = NULL,
  get.vnir = FALSE
)
```

Arguments

series	a soil series name; case-insensitive
bbox	a bounding box in WGS84 geographic coordinates e.g. `c(-120,37,-122,38)`, constrained to a 5-degree block
state	a two-letter US state abbreviation; case-insensitive
rcasiteid	a RaCA site id (e.g. 'C1609C01')
get.vnir	logical, should associated VNIR spectra be downloaded? (see details)

Details

The VNIR spectra associated with RaCA data are quite large [each gzip-compressed VNIR spectra record is about 6.6kb], so requests for these data are disabled by default. Note that VNIR spectra can only be queried by soil series or geographic BBOX.

Value

`pedons`: a `SoilProfileCollection` object containing site/pedon/horizon data

`trees`: a `data.frame` object containing tree DBH and height

`veg`: a `data.frame` object containing plant species

`stock`: a `data.frame` object containing carbon quantities (stocks) at standardized depths

`sample`: a `data.frame` object containing sample-level bulk density and soil organic carbon values

`spectra`: a numeric `matrix` containing VNIR reflectance spectra from 350–2500 nm

Author(s)

D.E. Beaudette, USDA-NRCS staff

References

https://www.nrcs.usda.gov/wps/portal/nrcs/detail/soils/survey/?cid=nrcs142p2_054164
fetchRaCA() Tutorial

See Also

fetchOSD

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

  if(require(aqp)) {

    # search by series name
    s <- fetchRaCA(series='auburn')

    # search by bounding-box
    # s <- fetchRaCA(bbox=c(-120, 37, -122, 38))

    # check structure
    str(s, 1)

    # extract pedons
    p <- s$pedons

    # how many pedons
    length(p)

    # plot
    par(mar=c(0,0,0,0))
    plot(p, name='hzn_desgn', max.depth=150)
  }
}
```

---

fetchSCAN *Fetch SCAN Data*

Description

Query soil/climate data from USDA-NRCS SCAN Stations (experimental)

Usage

```
# get SCAN data
fetchSCAN(site.code, year, report='SCAN', req=NULL)

# get sensor metadata for one or more sites
SCAN_sensor_metadata(site.code)

# get site metadata for one or more sites
SCAN_site_metadata(site.code)
```

Arguments

site.code	a vector of site codes
year	a vector of years
report	report name, single value only
req	list of SCAN request parameters, for backwards-compatibility only

Details

See the fetchSCAN tutorial for details. These functions require the 'httr' and 'rvest' libraries.

Value

a data.frame object

Note

SCAN_sensor_metadata() is known to crash on 32bit R / libraries (Windows).

Author(s)

D.E. Beaudette

References

https://www.wcc.nrcs.usda.gov/index.html

Examples

```
if(requireNamespace("curl") &
    curl::has_internet()) {

    # get data: new interface
    x <- fetchSCAN(site.code=c(356, 2072), year=c(2015, 2016))
    str(x)

    # get sensor metadata
    m <- SCAN_sensor_metadata(site.code=c(356, 2072))
```

```
    # get site metadata
    m <- SCAN_site_metadata(site.code=c(356, 2072))
}
```

---

fetchSDA *Download and Flatten Data from Soil Data Access*

Description

Functions to download and flatten commonly used tables and from Soil Data Access, and create soil profile collection objects (SPC).

Usage

```
fetchSDA(WHERE = NULL, duplicates = FALSE, childs = TRUE,
         nullFragsAreZero = TRUE, rmHzErrors = FALSE,
         droplevels = TRUE,
         stringsAsFactors = default.stringsAsFactors()
         )

get_mapunit_from_SDA(WHERE = NULL,
                     droplevels = TRUE,
                     stringsAsFactors = default.stringsAsFactors()
                     )

get_component_from_SDA(WHERE = NULL, duplicates = FALSE, childs = TRUE,
                       droplevels = TRUE, nullFragsAreZero = TRUE,
                       stringsAsFactors = default.stringsAsFactors()
                       )

get_chorizon_from_SDA(WHERE = NULL, duplicates = FALSE, childs = TRUE,
                      nullFragsAreZero = TRUE,
                      droplevels = TRUE,
                      stringsAsFactors = default.stringsAsFactors()
                      )

get_cosoilmoist_from_SDA(WHERE = NULL, duplicates = FALSE, impute = TRUE,
                         stringsAsFactors = default.stringsAsFactors()
                         )
```

Arguments

WHERE	text string formatted as an SQL WHERE clause (default: FALSE)
duplicates	logical; if TRUE a record is returned for each unique mukey (may be many per nationalmusym)

childs — logical; if FALSE parent material and geomorphic child tables are not flattened and appended

impute — replace missing (i.e. NULL) values with "Not_Populated" for categorical data, or the "RV" for numeric data or 201 cm if the "RV" is also NULL (default: TRUE)

nullFragsAreZero — should fragment volumes of NULL be interpreted as 0? (default: TRUE), see details

rmHzErrors — should pedons with horizonation errors be removed from the results? (default: FALSE)

droplevels — logical: indicating whether to drop unused levels in classifying factors. This is useful when a class has large number of unused classes, which can waste space in tables and figures.

stringsAsFactors — logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

These functions return data from Soil Data Access with the use of a simple text string that formatted as an SQL WHERE clause (e.g. `WHERE = "areasymbol = 'IN001'"`. All functions are SQL queries that wrap around `SDAquery()` and format the data for analysis.

Beware SDA includes the data for both SSURGO and STATSGO2. The `areasymbol` for STATSGO2 is `US`. For just SSURGO, include `WHERE = "areareasymbol != 'US'"`.

If the duplicates argument is set to TRUE, duplicate components are returned. This is not necessary with data returned from NASIS, which has one unique national map unit. SDA has duplicate map national map units, one for each legend it exists in.

The value of `nullFragsAreZero` will have a significant impact on the rock fragment fractions returned by `fetchSDA`. Set `nullFragsAreZero = FALSE` in those cases where there are many data-gaps and NULL rock fragment values should be interpreted as NULLs. Set `nullFragsAreZero = TRUE` in those cases where NULL rock fragment values should be interpreted as 0.

Value

A data.frame or SoilProfileCollection object.

Author(s)

Stephen Roecker

See Also

SDA_query

Examples

```
if (requireNamespace("curl") &
  curl::has_internet() &
  require(aqp) &
  require("ggplot2") &
  require("gridExtra") &
  require("viridis")
) {

  # query soil components by areasymbol and musym
  test = fetchSDA(WHERE = "areasymbol = 'IN005' AND musym = 'MnpB2'")


  # profile plot
  plot(test)


  # convert the data for depth plot
  clay_slice = horizons(slice(test, 0:200 ~ claytotal_l + claytotal_r + claytotal_h))
  names(clay_slice) <- gsub("claytotal_", "", names(clay_slice))

  om_slice = horizons(slice(test, 0:200 ~ om_l + om_r + om_h))
  names(om_slice) = gsub("om_", "", names(om_slice))

  test2 = rbind(data.frame(clay_slice, var = "clay"),
                data.frame(om_slice, var = "om")
  )

  h = merge(test2, site(test)[c("nationalmusym", "cokey", "compname", "comppct_r")],
            by = "cokey",
            all.x = TRUE
  )

  # depth plot of clay content by soil component
  gg_comp <- function(x) {
    ggplot(x) +
      geom_line(aes(y = r, x = hzdept_r)) +
      geom_line(aes(y = r, x = hzdept_r)) +
      geom_ribbon(aes(ymin = l, ymax = h, x = hzdept_r), alpha = 0.2) +
      xlim(200, 0) +
      xlab("depth (cm)") +
      facet_grid(var ~ nationalmusym + paste(compname, comppct_r)) +
      coord_flip()
  }
  g1 <- gg_comp(subset(h, var == "clay"))
  g2 <- gg_comp(subset(h, var == "om"))

  grid.arrange(g1, g2)
```

```
# query cosoilmoist (e.g. water table data) by mukey
x <- get_cosoilmoist_from_SDA(WHERE = "mukey = '1395352'")

ggplot(x, aes(x = as.integer(month), y = dept_r, lty = status)) +
  geom_rect(aes(xmin = as.integer(month), xmax = as.integer(month) + 1,
                ymin = 0, ymax = max(x$depb_r),
                fill = flodfreqcl)) +
  geom_line(cex = 1) +
  geom_point() +
  geom_ribbon(aes(ymin = dept_l, ymax = dept_h), alpha = 0.2) +
  ylim(max(x$depb_r), 0) +
  xlab("month") + ylab("depth (cm)") +
  scale_x_continuous(breaks = 1:12, labels = month.abb, name="Month") +
  facet_wrap(~ paste0(compname, ' (', comppct_r , ')')) +
  ggtitle(paste0(x$nationalmusym[1],
                 ': Water Table Levels from Component Soil Moisture Month Data'))



# query all Miami major components
s <- get_component_from_SDA(WHERE = "compname = 'Miami' \n
              AND majcompflag = 'Yes' AND areasymbol != 'US'")


# landform vs 3-D morphometry
test <- {
  subset(s, ! is.na(landform) | ! is.na(geompos)) ->.;
  split(., .$drainagecl, drop = TRUE) ->.;
  lapply(., function(x) {
    test = data.frame()
    test = as.data.frame(table(x$landform, x$geompos))
    test$compname   = x$compname[1]
    test$drainagecl = x$drainagecl[1]
    names(test)[1:2] <- c("landform", "geompos")
    return(test)
  }) ->.;
  do.call("rbind", .) ->.;
  .[.$Freq > 0, ] ->.;
  within(., {
    landform = reorder(factor(landform), Freq, max)
    geompos  = reorder(factor(geompos),  Freq, max)
    geompos  = factor(geompos, levels = rev(levels(geompos)))
  }) ->.;
}
test$Freq2 <- cut(test$Freq,
                  breaks = c(0, 5, 10, 25, 50, 100, 150),
                  labels = c("<5", "5-10", "10-25", "25-50", "50-100", "100-150")
)
ggplot(test, aes(x = geompos, y = landform, fill = Freq2)) +
  geom_tile(alpha = 0.5) + facet_wrap(~ paste0(compname, "\n", drainagecl)) +
  scale_fill_viridis(discrete = TRUE) +
 theme(aspect.ratio = 1, axis.text.x = element_text(angle = 45, hjust = 1, vjust = 1)) +
```

```
    ggtitle("Landform vs 3-D Morphometry for Miami Major Components on SDA")

}
```

`fetchSDA_spatial` *Query SDA and Return Spatial Data*

Description

This is a high-level "fetch" method to facilitate spatial queries to Soil Data Access (SDA) based on `mukey` or `nationalmusym`.

A SDA spatial query is made returning geometry and key identifying information about the mapunit. Additional columns from the mapunit table can be included using `add.fields` argument.

This function automatically "chunks" the input vector (using `soilDB::makeChunks`) of mapunit identifiers to minimize the likelihood of exceeding the SDA data request size. The number of chunks varies with the `chunk.size` setting and the length of your input vector. If you are working with many mapunits and/or large extents, you may need to decrease this number in order to have more chunks.

The "sweet-spot" `chunk.size` should optimize number of queries relative to the typical amount of information in each query. There is a 100,000 record limit per query and the type / complexity of the geometry affects the total result size; there is a ~32Mb JSON serialization limit.

Querying regions with complex mapping may require smaller `chunk.size`. Numerically adjacent IDs in the input vector may share common qualities (say, all from same soil survey area or region) which could cause specific chunks to perform "poorly" [slow or error] no matter what the chunk size is. Shuffling the order of the inputs using `sample` may help to eliminate problems related to this, depending on how you obtained your set of MUKEY/nationalmusym to query. One could feasibly use `muacres` as a heuristic to adjust for total acreage within chunks.

Usage

```
fetchSDA_spatial(
  x,
  by.col = "mukey",
  method = "feature",
  add.fields = NULL,
  chunk.size = 10,
  verbose = TRUE
)
```

Arguments

x	A vector of MUKEYs or national mapunit symbols.
by.col	Column name containing mapunit identifier ("mukey" or "nmusym"); default: "mukey"
method	geometry result type: 'feature' returns polygons, 'bbox' returns the bounding box of each polygon, and 'point' returns a single point within each polygon.
add.fields	Column names from 'mapunit' table to add to result. Must specify table name prefix 'mapunit' before column name (e.g. 'mapunit.muname').
chunk.size	How many queries should spatial request be divided into? Necessary for large results. Default: 10
verbose	Print messages?

Value

A Spatial*DataFrame corresponding to SDA spatial data for all MUKEYs / nmusyms requested. Default result contains mapunit delineation geometry with attribute table containing 'gid', 'mukey' and 'nationalmusym', plus additional fields in result specified with 'add.fields'.

Author(s)

Andrew G. Brown

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

   # get spatial data for a single mukey
    single.mukey <- fetchSDA_spatial(x = "2924882")

    # demonstrate fetching full extent (multi-mukey) of national musym
    full.extent.nmusym <- fetchSDA_spatial(x = "2x8l5", by = "nmusym")

    # compare extent of nmusym to single mukey within it
    if(require(sp)) {
     plot(full.extent.nmusym, col = "RED",border=0)
     plot(single.mukey, add = TRUE, col = "BLUE", border=0)
    }

    # demo adding a field (`muname`) to attribute table of result
    head(fetchSDA_spatial(x = "2x8l5", by="nmusym", add.fields="muname"))
}
```

`fetchSoilGrids` *Fetch SoilGrids 250m properties information from point locations*

Description

This function obtains SoilGrids properties information (250m raster resolution) given a `data.frame` containing site IDs, latitudes and longitudes.

Usage

```
fetchSoilGrids(locations, loc.names = c("id", "lat", "lon"))
```

Arguments

`locations`	A `data.frame` containing 3 columns referring to site ID, latitude and longitude.
`loc.names`	Optional: Column names referring to site ID, latitude and longitude. Default: `c("id","lat","lon")`

Details

The depth intervals returned are: `"0-5cm","5-15cm","15-30cm","30-60cm","60-100cm","100-200cm"` and the properties returned are `"bdod","cec","cfvo","clay","nitrogen","phh2o","sand","silt","soc"` – each with 5th, 50th, 95th, mean and uncertainty values. Point data requests are made through `properties/query` endpoint of the SoilGrids v2.0 REST API: https://rest.soilgrids.org/soilgrids/v2.0/docs

Value

A SoilProfileCollection

Author(s)

Andrew G. Brown

Examples

```
if(requireNamespace("curl") &
  curl::has_internet()) {

 library(aqp)

 your.points <- data.frame(id  = c("A", "B"),
                           lat = c(37.9, 38.1),
                           lon = c(-120.3, -121.5),
                           stringsAsFactors = FALSE)

 x <- fetchSoilGrids(your.points)

 plotSPC(x, name = NA, color = "socQ50")
```

```
}
```

`filter_geochem` *Filter KSSL Geochemical Table*

Description

A function to subset KSSL "geochem" / elemental analysis result table to obtain rows/columns based on: column name, preparation code, major / trace element method.

Usage

```
filter_geochem(
  geochem,
  columns = NULL,
  prep_code = NULL,
  major_element_method = NULL,
  trace_element_method = NULL
)
```

Arguments

`geochem`	geochemical data, as returned by fetchKSSL
`columns`	Column name(s) to include in result
`prep_code`	Character vector of prep code(s) to include in result.
`major_element_method`	Character vector of major element method(s) to include in result.
`trace_element_method`	Character vector of trace element method(s) to include in result.

Value

A data.frame, subsetted according to the constraints specified in arguments.

Author(s)

Andrew G. Brown.

`format_SQL_in_statement` *Format vector of values into a string suitable for an SQL 'IN' statement.*

Description

Concatenate a vector to SQL `IN`-compatible syntax: `letters[1:3]` becomes `('a','b','c')`. Values in `x` are first passed through `unique()`.

Usage

```
format_SQL_in_statement(x)
```

Arguments

`x`	A character vector.

Value

A character vector (unit length) containing concatenated group syntax for use in SQL `IN`, with unique value found in `x`.

Note

Only `character` output is supported.

Examples

```
library(aqp)

# get some mukeys
q <- "select top(2) mukey from mapunit;"
mukeys <- SDA_query(q)

# format for use in an SQL IN statement
mukey.inst <- format_SQL_in_statement(mukeys$mukey)
mukey.inst

# make a more specific query: for component+horizon data, just for those mukeys
q2 <- sprintf("SELECT * FROM mapunit
               INNER JOIN component ON mapunit.mukey = component.mukey
               INNER JOIN chorizon ON component.cokey = chorizon.cokey
               WHERE mapunit.mukey IN %s;", mukey.inst)
# do the query
res <- SDA_query(q2)
```

```
# build a SoilProfileCollection from horizon-level records
depths(res) <- cokey ~ hzdept_r + hzdepb_r

# normalize mapunit/component level attributes to site-level for plot
site(res) <- ~ muname + mukey + compname + comppct_r + taxclname

# make a nice label
res$labelname <- sprintf("%s (%s%s)", res$compname, res$comppct_r, "%")

# major components only
res <- filter(res, comppct_r >= 85)

# inspect plot of result
par(mar=c(0,0,0,0))
groupedProfilePlot(res, groups = "mukey", color = "hzname", cex.names=0.8,
                   id.style = "side", label = "labelname")
```

`get_colors_from_NASIS_db` *Extract Soil Color Data from a local NASIS Database*

Description

Get, format, mix, and return color data from a NASIS database.

Usage

```
get_colors_from_NASIS_db(SS = TRUE)
```

Arguments

`SS`	fetch data from Selected Set in NASIS or from the entire local database (default: TRUE)

Details

This function currently works only on Windows.

Value

A data.frame with the results.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

simplifyColorData, get_hz_data_from_NASIS_db, get_site_data_from_NASIS_db

---

get_colors_from_pedon_db *Extract Soil Color Data from a PedonPC Database*

Description

Get, format, mix, and return color data from a PedonPC database.

Usage

```
get_colors_from_pedon_db(dsn)
```

Arguments

dsn	The path to a 'pedon.mdb' database.

Details

This function currently works only on Windows.

Value

A data.frame with the results.

Author(s)

Dylan E. Beaudette and Jay M. Skovlin

See Also

get_hz_data_from_pedon_db, get_site_data_from_pedon_db

get_comonth_from_NASIS_db *Extract component month data from a local NASIS Database*

Description

Extract component month data from a local NASIS Database.

Usage

```
get_comonth_from_NASIS_db(SS = TRUE, fill = FALSE,
                          stringsAsFactors = default.stringsAsFactors()
                          )
```

Arguments

SS	get data from the currently loaded Selected Set in NASIS or from the entire local database (default: TRUE)
fill	should missing "month" rows in the comonth table be filled with NA (FALSE)
stringsAsFactors	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Stephen Roecker

See Also

fetchNASIS

Examples

```
if(local_NASIS_defined()) {
  # query text note data
  cm <- try(get_comonth_from_NASIS_db())

  # show structure of component month data
  str(cm)
}
```

`get_component_data_from_NASIS_db` *Extract component data from a local NASIS Database*

Description

Extract component data from a local NASIS Database.

Usage

```
get_component_data_from_NASIS_db(SS = TRUE, stringsAsFactors = default.stringsAsFactors())
get_component_restrictions_from_NASIS_db(SS = TRUE)
```

Arguments

`SS`	get data from the currently loaded Selected Set in NASIS or from the entire local database (default: TRUE)
`stringsAsFactors`	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Dylan E. Beaudette, Stephen Roecker, and Jay M. Skovlin

See Also

`fetchNASIS`

Examples

```
if(local_NASIS_defined()) {
 # query text note data
 fc <- try(get_component_data_from_NASIS_db())

 # show structure of component data returned
 str(fc)
}
```

`get_cosoilmoist_from_NASIS` *Read and Flatten the Component Soil Moisture Tables*

Description

Read and flatten the component soil moisture month tables from a local NASIS Database.

Usage

```
get_cosoilmoist_from_NASIS(impute = TRUE, stringsAsFactors = default.stringsAsFactors())
```

Arguments

`impute`	replace missing (i.e. NULL) values with "Not_Populated" for categorical data, or the "RV" for numeric data or 201 cm if the "RV" is also NULL (default: TRUE)
`stringsAsFactors`	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

The component soil moisture tables within NASIS house monthly data on flooding, ponding, and soil moisture status. The soil moisture status is used to specify the water table depth for components (e.g. `status == "Moist"`).

Value

A data.frame.

Note

This function currently works only on Windows.

Author(s)

S.M. Roecker

See Also

fetchNASIS, get_cosoilmoist_from_NASISWebReport, get_cosoilmoist_from_SDA, `get_comonth_from_SDA`

Examples

```
if(local_NASIS_defined()) {
 # load cosoilmoist (e.g. water table data)
 test <- try(get_cosoilmoist_from_NASIS())

 # inspect
 if(!inherits(test, 'try-error')) {
   head(test)
 }
}
```

`get_extended_data_from_NASIS_db`
Extract accessory tables and summaries from a local NASIS Database

---

Description

Extract accessory tables and summaries from a local NASIS Database.

Usage

```
get_extended_data_from_NASIS_db(SS = TRUE, nullFragsAreZero = TRUE,
                                stringsAsFactors = default.stringsAsFactors()
                                )
```

Arguments

`SS`	get data from the currently loaded Selected Set in NASIS or from the entire local database (default: TRUE)
`nullFragsAreZero`	should fragment volumes of NULL be interpreted as 0? (default: TRUE), see details
`stringsAsFactors`	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have been set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

get_hz_data_from_NASIS_db, get_site_data_from_NASIS_db

Examples

```
if(local_NASIS_defined()) {
 # query extended data
 e <- try(get_extended_data_from_NASIS_db())

 # show contents of extended data
 str(e)
}
```

---

get_extended_data_from_pedon_db

Extract accessory tables and summaries from a local pedonPC Database

---

Description

Extract accessory tables and summaries from a local pedonPC Database.

Usage

```
get_extended_data_from_pedon_db(dsn)
```

Arguments

dsn	The path to a 'pedon.mdb' database.

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

get_hz_data_from_pedon_db, get_site_data_from_pedon_db

`get_hz_data_from_NASIS_db` *Extract Horizon Data from a local NASIS Database*

Description

Get horizon-level data from a local NASIS database.

Usage

```
get_hz_data_from_NASIS_db(SS = TRUE, stringsAsFactors = default.stringsAsFactors())
```

Arguments

SS	fetch data from Selected Set in NASIS or from the entire local database (default: TRUE)
stringsAsFactors	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have been set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

This function currently works only on Windows.

Value

A data.frame.

Note

NULL total rock fragment values are assumed to represent an _absence_ of rock fragments, and set to 0.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

get_hz_data_from_NASIS_db, get_site_data_from_NASIS_db

---

get_hz_data_from_pedon_db

Extract Horizon Data from a PedonPC Database

Description

Get horizon-level data from a PedonPC database.

Usage

```
get_hz_data_from_pedon_db(dsn)
```

Arguments

dsn	The path to a 'pedon.mdb' database.

Details

This function currently works only on Windows.

Value

A data.frame.

Note

NULL total rock fragment values are assumed to represent an _absence_ of rock fragments, and set to 0.

Author(s)

Dylan E. Beaudette and Jay M. Skovlin

See Also

get_colors_from_pedon_db, get_site_data_from_pedon_db

`get_lablayer_data_from_NASIS_db` *Extract lab pedon layer data from a local NASIS Database*

Description

Get lab pedon layer-level(horizon-level) data from a local NASIS database.

Usage

```
get_lablayer_data_from_NASIS_db(SS = TRUE)
```

Arguments

SS	fetch data from the currently loaded selected set in NASIS or from the entire local database (default: TRUE)

Details

This function currently works only on Windows, and requires a 'nasis_local' ODBC connection.

Value

A data.frame.

Note

This function queries KSSL laboratory site/horizon data from a local NASIS database from the lab layer data table.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

`get_labpedon_data_from_NASIS_db`

`get_labpedon_data_from_NASIS_db` *Extract lab pedon data from a local NASIS Database*

Description

Get lab pedon-level data from a local NASIS database.

Usage

```
get_labpedon_data_from_NASIS_db(SS = TRUE)
```

Arguments

SS	fetch data from the currently loaded selected set in NASIS or from the entire local database (default: TRUE)

Details

This function currently works only on Windows, and requires a 'nasis_local' ODBC connection.

Value

A data.frame.

Note

This function queries KSSL laboratory site/horizon data from a local NASIS database from the lab pedon data table.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

`get_lablayer_data_from_NASIS_db`

get_NOAA_GHCND *Get Global Historical Climatology Network Daily (GHCND) data from NOAA API for given datatype(s), station IDs and years.*

Description

Obtain daily climatic summary data for a set of station IDs, years, and datatypes.

Note that typically results from the NOAA API are limited to 1000 records. However, by "chunking" up data into individual station*year*datatypeid combinations, record results generally do not exceed 365 records for daily summaries.

In order to use this function, you must obtain an API token from this website: https://www.ncdc.noaa.gov/cdo-web/token

Usage

```
get_NOAA_GHCND(stations, years, datatypeids, apitoken)
```

Arguments

stations	Station ID (e.g. GHCND:USC00388786)
years	One or more years (e.g. 2017:2020)
datatypeids	One or more NOAA GHCND data type IDs (e.g c("PRCP","SNOW"))
apitoken	API key token for NOAA NCDC web services (https://www.ncdc.noaa.gov/cdo-web/token)

Value

A data.frame containing the GHCND data requested (limit 1000 records)

Examples

```
#' ## in order to use this function, you must obtain an API token from this website:
##  https://www.ncdc.noaa.gov/cdo-web/token

# get_NOAA_GHCND(c("GHCND:USC00388786", "GHCND:USC00388787"),
#                years = 2017:2020,
#                datatypeids = c("PRCP","SNOW"),
#                apitoken = "yourtokenhere")
```

get_NOAA_stations_nearXY *Query the NOAA API to get station data near a given latitude and longitude*

Description

Query the NOAA API to get station data (limit 1000 records) near a point. Default extent is plus or minus 0.5 degrees (bounding box) (with `bbox = 1`) around the specified point [lat, lng].

In order to use this function, you must obtain an API token from this website: https://www.ncdc.noaa.gov/cdo-web/token

Usage

```
get_NOAA_stations_nearXY(lat, lng, apitoken, bbox = 1)
```

Arguments

`lat`	Latitude
`lng`	Longitude
`apitoken`	API key token for NOAA NCDC web service
`bbox`	Optional: Dimension of the bounding box centered at `lat`, `lng`.

Value

data.frame containing station information for all stations within a bounding box around `lat`, `lng`.

Examples

```
## in order to use this function, you must obtain an API token from this website:
##  https://www.ncdc.noaa.gov/cdo-web/token

# stations <- get_NOAA_stations_nearXY(lat = 37, lng = -120,
#                                      apitoken = "yourtokenhere")
```

`get_site_data_from_NASIS_db` *Extract Site Data from a local NASIS Database*

Description

Get site-level data from a local NASIS database.

Usage

```
get_site_data_from_NASIS_db(SS = TRUE, stringsAsFactors = default.stringsAsFactors())
```

Arguments

`SS`	fetch data from Selected Set in NASIS or from the entire local database (default: TRUE)
`stringsAsFactors`	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have been set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Details

When multiple "site bedrock" entries are present, only the shallowest is returned by this function.

Value

A data.frame.

Note

This function currently works only on Windows.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

See Also

`get_hz_data_from_NASIS_db`,

get_site_data_from_pedon_db

Extract Site Data from a PedonPC Database

Description

Get site-level data from a PedonPC database.

Usage

```
get_site_data_from_pedon_db(dsn)
```

Arguments

dsn	The path to a 'pedon.mdb' database.

Value

A data.frame.

Note

This function currently works only on Windows.

Author(s)

Dylan E. Beaudette and Jay M. Skovlin

See Also

get_hz_data_from_pedon_db, get_veg_from_AK_Site,

get_soilseries_from_NASIS

Get records from the Soil Classification (SC) database

Description

These functions return records from the Soil Classification database, either from the local NASIS database (all series) or via web report (named series only).

Usage

```
get_soilseries_from_NASIS(stringsAsFactors = default.stringsAsFactors())
get_soilseries_from_NASISWebReport(soils,
stringsAsFactors = default.stringsAsFactors())
```

Arguments

soils	character vector of soil series names
stringsAsFactors	logical: should character vectors be converted to factors? This argument is passed to the uncode() function. It does not convert those vectors that have set outside of uncode() (i.e. hard coded). The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)

Value

A data.frame.

Author(s)

Stephen Roecker

---

get_text_notes_from_NASIS_db

Extract text note data from a local NASIS Database

Description

Extract text note data from a local NASIS Database.

Usage

```
get_text_notes_from_NASIS_db(SS = TRUE, fixLineEndings = TRUE)
```

Arguments

SS	get data from the currently loaded Selected Set in NASIS or from the entire local database (default: TRUE)
fixLineEndings	convert line endings from "\r\n" to "\n"

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Dylan E. Beaudette and Jay M. Skovlin

See Also

get_hz_data_from_pedon_db, get_site_data_from_pedon_db

Examples

```
if(local_NASIS_defined()) {
 # query text note data
 t <- try(get_text_notes_from_NASIS_db())

 # show contents text note data, includes: siteobs, site, pedon, horizon level text notes data.
 str(t)

 # view text categories for site text notes
 if(!inherits(t, 'try-error')) {
  table(t$site_text$textcat)
 }
}
```

---

get_veg_data_from_NASIS_db

Extract veg data from a local NASIS Database

Description

Extract veg data from a local NASIS Database.

Usage

```
get_veg_data_from_NASIS_db(SS = TRUE)
```

Arguments

SS	get data from the currently loaded Selected Set in NASIS or from the entire local database (default: TRUE)

Details

This function currently works only on Windows.

Value

A list with the results.

Author(s)

Jay M. Skovlin and Dylan E. Beaudette

Examples

```
if(local_NASIS_defined()) {
 # query text note data
 v <- try(get_veg_from_NASIS_db())

 # show contents veg data returned
 str(v)
}
```

`get_veg_from_AK_Site` *Retrieve Vegetation Data from an AK Site Database*

Description

Retrieve Vegetation Data from an AK Site Database

Usage

```
get_veg_from_AK_Site(dsn)
```

Arguments

`dsn`	file path the the AK Site access database

Value

A data.frame with vegetation data in long format, linked to site ID.

Note

This function currently works only on Windows.

Author(s)

Dylan E. Beaudette

See Also

`get_hz_data_from_pedon_db`, `get_site_data_from_pedon_db`

get_veg_from_MT_veg_db

Extract Site and Plot-level Data from a Montana RangeDB database

Description

Get Site and Plot-level data from a Montana RangeDB database.

Usage

```
get_veg_from_MT_veg_db(dsn)
```

Arguments

dsn	The name of the Montana RangeDB front-end database connection (see details).

Details

This function currently works only on Windows.

Value

A data.frame.

Author(s)

Jay M. Skovlin

See Also

get_veg_species_from_MT_veg_db, get_veg_other_from_MT_veg_db

get_veg_from_NPS_PLOTS_db

Retrieve Vegetation Data from an NPS PLOTS Database

Description

Used to extract species, stratum, and cover vegetation data from a backend NPS PLOTS Database. Currently works for any Microsoft Access database with an .mdb file format.

Usage

```
get_veg_from_NPS_PLOTS_db(dsn)
```

Arguments

dsn	file path to the NPS PLOTS access database on your system.

Value

A data.frame with vegetation data in a long format with linkage to NRCS soil pedon data via the site_id key field.

Note

This function currently only works on Windows.

Author(s)

Jay M. Skovlin

---

get_veg_other_from_MT_veg_db

Extract cover composition data from a Montana RangeDB database

---

Description

Get cover composition data from a Montana RangeDB database.

Usage

```
get_veg_other_from_MT_veg_db(dsn)
```

Arguments

dsn	The name of the Montana RangeDB front-end database connection (see details).

Details

This function currently works only on Windows.

Value

A data.frame.

Author(s)

Jay M. Skovlin

See Also

get_veg_from_MT_veg_db, get_veg_species_from_MT_veg_db

get_veg_species_from_MT_veg_db *Extract species-level Data from a Montana RangeDB database*

Description

Get species-level data from a Montana RangeDB database.

Usage

```
get_veg_species_from_MT_veg_db(dsn)
```

Arguments

dsn	The name of the Montana RangeDB front-end database connection (see details).

Details

This function currently works only on Windows.

Value

A data.frame.

Author(s)

Jay M. Skovlin

See Also

get_veg_from_MT_veg_db, get_veg_other_from_MT_veg_db

KSSL_VG_model *Develop a Water Retention Curve from KSSL Data*

Description

Water retention curve modeling via van Genuchten model and KSSL data.

Usage

```
KSSL_VG_model(VG_params, phi_min = 10^-6, phi_max = 10^8, pts = 100)
```

Arguments

VG_params	a data.frame or list object with the parameters of the van Genuchten model, see details
phi_min	lower limit for water potential in kPa
phi_max	upper limit for water potential in kPa
pts	number of points to include in estimated water retention curve

Details

This function was developed to work with measured or estimated parameters of the van Genuchten model, as generated by the Rosetta model. As such, VG_params should have the following format and conventions:

theta_r saturated water content, values should be in the range of {0, 1}

theta_s residual water content, values should be in the range of {0, 1}

alpha related to the inverse of the air entry suction, function expects log10-transformed values with units of cm

npar index of pore size distribution, function expects log10-transformed values with units of 1/cm

Value

A list with the following components:

VG_curve estimated water retention curve: paired estimates of water potential (phi) and water content (theta)

VG_function spline function for converting water potential (phi, units of kPa) to estimated volumetric water content (theta, units of percent, range: {0, 1})

VG_inverse_function spline function for converting volumetric water content (theta, units of percent, range: {0, 1}) to estimated water potential (phi, units of kPa)

Note

A practical example is given in the fetchSCAN tutorial.

Author(s)

D.E. Beaudette

References

water retention curve estimation

van Genuchten, M.Th. (1980). "A closed-form equation for predicting the hydraulic conductivity of unsaturated soils". Soil Science Society of America Journal. 44 (5): 892-898.

Examples

```
# basic example
d <- data.frame(theta_r=0.0337216,
theta_s=0.4864061,
alpha=-1.581517,
npar=0.1227247)

vg <- KSSL_VG_model(d)

str(vg)
```

---

loafercreek *Example* SoilProfilecollection *Objects Returned by* fetchNASIS.

Description

Several examples of soil profile collections returned by `fetchNASIS(from='pedons')` as `SoilProfileCollection` objects.

Usage

```
data(loafercreek)
data(gopheridge)
data(mineralKing)
```

Examples

```
if(require("aqp")) {
# load example dataset
  data("gopheridge")

  # what kind of object is this?
  class(gopheridge)

  # how many profiles?
  length(gopheridge)

  # there are 60 profiles, this calls for a split plot
  par(mar=c(0,0,0,0), mfrow=c(2,1))

  # plot soil colors
  plot(gopheridge[1:30, ], name='hzname', color='soil_color')
  plot(gopheridge[31:60, ], name='hzname', color='soil_color')

  # need a larger top margin for legend
  par(mar=c(0,0,4,0), mfrow=c(2,1))
  # generate colors based on clay content
  plot(gopheridge[1:30, ], name='hzname', color='clay')
```

```
 plot(gopheridge[31:60, ], name='hzname', color='clay')

 # single row and no labels
 par(mar=c(0,0,0,0), mfrow=c(1,1))
 # plot soils sorted by depth to contact
 plot(gopheridge, name='', print.id=FALSE, plot.order=order(gopheridge$bedrckdepth))

 # plot first 10 profiles
plot(gopheridge[1:10, ], name='hzname', color='soil_color', label='pedon_id', id.style='side')

 # add rock fragment data to plot:
 addVolumeFraction(gopheridge[1:10, ], colname='total_frags_pct')

 # add diagnostic horizons
addDiagnosticBracket(gopheridge[1:10, ], kind='argillic horizon', col='red', offset=-0.4)

 ## loafercreek
 data("loafercreek")
 # plot first 10 profiles
plot(loafercreek[1:10, ], name='hzname', color='soil_color', label='pedon_id', id.style='side')

 # add rock fragment data to plot:
 addVolumeFraction(loafercreek[1:10, ], colname='total_frags_pct')

 # add diagnostic horizons
addDiagnosticBracket(loafercreek[1:10, ], kind='argillic horizon', col='red', offset=-0.4)
}
```

`local_NASIS_defined` *Check for presence of 'nasis_local' ODBC data source*

Description

Check for presence of 'nasis_local' ODBC data source

Usage

```
local_NASIS_defined()
```

Value

logical

Examples

```
if(local_NASIS_defined()) {
  # use fetchNASIS or some other lower-level fetch function
} else {
```

```
  message('could not find `nasis_local` ODBC data source')
}
```

---

makeChunks *Generate chunk labels for splitting data*

Description

Generate chunk labels for splitting data

Usage

```
makeChunks(ids, size = 100)
```

Arguments

ids	vector of IDs
size	chunk (group) size

Value

A numeric vector

Examples

```
# split the lowercase alphabet into 2 chunks

aggregate(letters,
          by = list(makeChunks(letters, size=13)),
          FUN = paste0, collapse=",")
```

---

mapunit_geom_by_ll_bbox
Fetch Map Unit Geometry from SDA

Description

Fetch map unit geometry from the SDA website by WGS84 bounding box. There is a limit on the amount of data returned as serialized JSON (~32Mb) and a total record limit of 100,000.

Usage

```
mapunit_geom_by_ll_bbox(bbox, source = "sda")
```

Arguments

bbox	a bounding box in WGS coordinates
source	the source database, currently limited to soil data access (SDA)

Details

The SDA website can be found at https://sdmdataaccess.nrcs.usda.gov. See examples for bounding box formatting.

Value

A SpatialPolygonsDataFrame of map unit polygons, in WGS84 (long,lat) coordinates.

Note

SDA does not return the spatial intersection of map unit polygons and bounding box. Rather, just those polygons that are completely within the bounding box / overlap with the bbox. This function requires the 'rgdal' package.

Author(s)

Dylan E. Beaudette

Examples

```
## fetch map unit geometry from a bounding-box:
#
#         +------------- (-120.41, 38.70)
#         |                     |
#         |                     |
# (-120.54, 38.61) --------------+
#

if(requireNamespace("curl") &
curl::has_internet() &
  require(sp) &
  require(rgdal)) {

    # basic usage
    b <- c(-120.54,38.61,-120.41,38.70)
    x <- try(mapunit_geom_by_ll_bbox(b)) # about 20 seconds

    if(!inherits(x,'try-error')) {
      # note that the returned geometry is everything overlapping the bbox
      # and not an intersection... why?
      plot(x)
    rect(b[1], b[2], b[3], b[4], border='red', lwd=2)


    # get map unit data for matching map unit keys
    in.statement <- format_SQL_in_statement(unique(x$mukey))
```

```
  q <- paste("SELECT mukey, muname FROM mapunit WHERE mukey IN ", in.statement, sep="")
  res <- SDA_query(q)

  #inspect
  head(res)
} else {
  message('could not download XML result from SDA')
}
}
```

OSDquery *Full text searching of the USDA-NRCS Official Series Descriptions*

Description

This is a rough example of how chunks of text parsed from OSD records can be made search-able with the PostgreSQL fulltext indexing and query system (syntax details). Each search field (except for the "brief narrative" and MLRA) corresponds with a section header in an OSD. The results may not include every OSD due to formatting errors and typos. Results are scored based on the number of times search terms match words in associated sections. This is the R API corresponding to this webpage.

Usage

```
OSDquery(mlra='', taxonomic_class='', typical_pedon='',
brief_narrative='', ric='', use_and_veg='',
competing_series='', geog_location='', geog_assoc_soils='')
```

Arguments

`mlra`	a comma-delimited list of MLRA to search
`taxonomic_class`	search family level classification
`typical_pedon`	search typical pedon section
`brief_narrative`	search brief narrative
`ric`	search range in characteristics section
`use_and_veg`	search use and vegetation section
`competing_series`	search competing section
`geog_location`	search geographic setting section
`geog_assoc_soils`	search geographically associated soils section

Details

See this webpage for more information.

family level taxa are derived from SC database, not parsed OSD records

MLRA are derived via spatial intersection (SSURGO x MLRA polygons)

MLRA-filtering is only possible for series used in the current SSURGO snapshot (component name)

logical AND: &

logical OR: |

wildcard, e.g. rhy-something rhy:*

search terms with spaces need doubled single quotes: ”san joaquin”

combine search terms into a single expression: (grano:* | granite)

Related documentation can be found in the following tutorials

- overview of all soil series query functions
- competing soil series
- siblings

Value

a `data.frame` object containing soil series names that match patterns supplied as arguments.

Note

SoilWeb maintains a snapshot of the Official Series Description data.

Author(s)

D.E. Beaudette

References

https://www.nrcs.usda.gov/wps/portal/nrcs/detailfull/soils/home/?cid=nrcs142p2_053587

See Also

fetchOSD

Examples

```
if(requireNamespace("curl") &
    curl::has_internet() &
    require(aqp)) {

  # find all series that list Pardee as a geographically associated soil.
  s <- OSDquery(geog_assoc_soils = 'pardee')
```

```
  # get data for these series
  x <- fetchOSD(s$series, extended = TRUE, colorState = 'dry')

  # simple figure
  par(mar=c(0,0,1,1))
  plot(x$SPC)
}
```

`parseWebReport` *Parse contents of a web report, based on supplied arguments.*

Description

Parse contents of a web report, based on supplied arguments.

Usage

```
parseWebReport(url, args, index = 1)
```

Arguments

`url`	Base URL to a LIMS/NASIS web report.
`args`	List of named arguments to send to report, see details.
`index`	Integer index specifying the table to return, or, NULL for a list of tables

Details

Report argument names can be inferred by inspection of the HTML source associated with any given web report.

Value

A `data.frame` object in the case of a single integer passed to `index`, a `list` object in the case of an integer vector or NULL passed to `index`.

Note

Most web reports are for internal use only.

Author(s)

D.E. Beaudette and S.M. Roecker

Examples

```
# pending
```

processSDA_WKT *Post-process WKT returned from SDA.*

Description

This is a helper function, commonly used with SDA_query to extract WKT (well-known text) representation of geometry to an sp-class object.

Usage

```
processSDA_WKT(d, g = "geom", p4s = "+proj=longlat +datum=WGS84")
```

Arguments

d	data.frame returned by SDA_query, containing WKT representation of geometry
g	name of column in d containing WKT geometry
p4s	PROJ4 CRS definition, typically GCS WGS84

Details

The SDA website can be found at https://sdmdataaccess.nrcs.usda.gov. See the SDA Tutorial for detailed examples.

Value

A Spatial* object.

Note

This function requires the 'httr', 'jsonlite', 'XML', and 'rgeos' packages.

Author(s)

D.E. Beaudette

SCAN_SNOTEL_metadata *SCAN and SNOTEL Station Metadata*

Description

SCAN and SNOTEL station metadata, a work in progress.

Usage

```
data("SCAN_SNOTEL_metadata")
```

Format

A data frame with 1092 observations on the following 12 variables.

`Name` station name

`Site` station ID

`State` state

`Network` sensor network: SCAN / SNOTEL

`County` county

`Elevation_ft` station elevation in feet

`Latitude` latitude of station

`Longitude` longitude of station

`HUC` associated watershed

`climstanm` climate station name (TODO: remove this column)

`upedonid` associated user pedon ID

`pedlabsampnum` associated lab sample ID

Details

These data have been compiled from several sources and represent a progressive effort to organize SCAN/SNOTEL station metadata. Therefore, some records may be missing or incorrect. Details on this effort can be found at the associated GitHub issue page: https://github.com/ncss-tech/soilDB/issues/61.

SDA_query *Soil Data Access Query*

Description

Submit a query to the Soil Data Access (SDA) REST/JSON web-service and return the results as a data.frame. There is a 100,000 record limit and 32Mb JSON serializer limit, per query. Queries should contain a WHERE statement or JOIN condition to limit the number of rows affected / returned. Consider wrapping calls to SDA_query in a function that can iterate over logical chunks (e.g. areasymbol, mukey, cokey, etc.). The function makeChunks can help with such iteration.

Usage

```
SDA_query(q)
```

Arguments

q	A valid T-SQL query surrounded by double quotes

Details

The SDA website can be found at https://sdmdataaccess.nrcs.usda.gov and query examples can be found at https://sdmdataaccess.nrcs.usda.gov/QueryHelp.aspx. A library of query examples can be found at https://nasis.sc.egov.usda.gov/NasisReportsWebSite/limsreport.aspx?report_name=SDA-SQL_Library_Home.

SSURGO (detailed soil survey) and STATSGO (generalized soil survey) data are stored together within SDA. This means that queries that don't specify an area symbol may result in a mixture of SSURGO and STATSGO records. See the examples below and the SDA Tutorial for details.

Value

a data.frame result (NULL if empty, try-error on error)

Note

This function requires the 'httr', 'jsonlite', and 'XML' packages

Author(s)

D.E. Beaudette

See Also

mapunit_geom_by_ll_bbox

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

  ## get SSURGO export date for all soil survey areas in California
  # there is no need to filter STATSGO
  # because we are filtering on SSURGO areasymbols
  q <- "SELECT areasymbol, saverest FROM sacatalog WHERE areasymbol LIKE 'CA%';"
  x <- SDA_query(q)
  head(x)


  ## get SSURGO component data associated with the
  ## Amador series / major component only
  # this query must explicitly filter out STATSGO data
  q <- "SELECT cokey, compname, comppct_r FROM legend
    INNER JOIN mapunit mu ON mu.lkey = legend.lkey
    INNER JOIN component co ON mu.mukey = co.mukey
    WHERE legend.areasymbol != 'US' AND compname = 'Amador';"

  res <- SDA_query(q)
  str(res)


  ## get component-level data for a specific soil survey area (Yolo county, CA)
  # there is no need to filter STATSGO because the query contains
  # an implicit selection of SSURGO data by areasymbol
  q <- "SELECT
    component.mukey, cokey, comppct_r, compname, taxclname,
    taxorder, taxsuborder, taxgrtgroup, taxsubgrp
    FROM legend
    INNER JOIN mapunit ON mapunit.lkey = legend.lkey
    LEFT OUTER JOIN component ON component.mukey = mapunit.mukey
    WHERE legend.areasymbol = 'CA113' ;"

  res <- SDA_query(q)
  str(res)

  ## get tabular data based on result from spatial query
  # there is no need to filter STATSGO because
  # SDA_Get_Mukey_from_intersection_with_WktWgs84() implies SSURGO
  #
  # requires raster and rgeos packages because raster is suggested
  # and rgeos is additional
  if(require(raster) & require(rgeos)) {
    # text -> bbox -> WKT
    # xmin, xmax, ymin, ymax
    b <- c(-120.9, -120.8, 37.7, 37.8)
    p <- writeWKT(as(extent(b), 'SpatialPolygons'))
    q <- paste0("SELECT mukey, cokey, compname, comppct_r FROM component
      WHERE mukey IN (SELECT DISTINCT mukey FROM
```

```
      SDA_Get_Mukey_from_intersection_with_WktWgs84('", p,
       "')) ORDER BY mukey, cokey, comppct_r DESC")

    x <- SDA_query(q)
    str(x)
  }
}
```

SDA_spatialQuery *SDA Spatial Query*

Description

Query SDA (SSURGO / STATSGO) records via spatial intersection with supplied geometries. Input can be SpatialPoints, SpatialLines, or SpatialPolygons objects with a valid CRS. Map unit keys, overlapping polygons, or the spatial intersection of `geom` + SSURGO / STATSGO polygons can be returned. See details.

Usage

```
SDA_spatialQuery(
  geom,
  what = "mukey",
  geomIntersection = FALSE,
  db = c("SSURGO", "STATSGO")
)
```

Arguments

`geom`	a Spatial* object, with valid CRS. May contain multiple features.
`what`	a character vector specifying what to return. 'mukey': `data.frame` with intersecting map unit keys and names, `geom` overlapping or intersecting map unit polygons
`geomIntersection`	logical; `FALSE`: overlapping map unit polygons returned, `TRUE`: intersection of `geom` + map unit polygons is returned.
`db`	a character vector identifying the Soil Geographic Databases ('SSURGO' or 'STATSGO') to query. Option *STATSGO* currently works only in combination with `what = "geom"`.

Details

Queries for map unit keys are always more efficient vs. queries for overlapping or intersecting (i.e. least efficient) features. `geom` is converted to GCS / WGS84 as needed. Map unit keys are always returned when using `what = "geom"`.

There is a 100,000 record limit and 32Mb JSON serializer limit, per query.

SSURGO (detailed soil survey, typically 1:24,000 scale) and STATSGO (generalized soil survey, 1:250,000 scale) data are stored together within SDA. This means that queries that don't specify an area symbol may result in a mixture of SSURGO and STATSGO records. See the examples below and the SDA Tutorial for details.

Value

A `data.frame` if `what = 'mukey'`, otherwise `SpatialPolygonsDataFrame` object.

Note

Row-order is not preserved across features in `geom` and returned object. Use `sp::over()` or similar functionality to extract from results. Polygon area in acres is computed server-side when `what = 'geom'` and `geomIntersection = TRUE`.

Author(s)

D.E. Beaudette, A.G. Brown, D.R. Schlaepfer

See Also

SDA_query

Examples

```
if(requireNamespace("curl") &
   curl::has_internet() &
   requireNamespace("sp") &
   requireNamespace("raster")
   ) {

library(aqp)
library(sp)
library(raster)

## query at a point

# example point
p <- SpatialPoints(cbind(x = -119.72330, y = 36.92204),
                   proj4string = CRS('+proj=longlat +datum=WGS84'))

# query map unit records at this point
res <- SDA_spatialQuery(p, what = 'mukey')

# convert results into an SQL "IN" statement
# useful when there are multiple intersecting records
mu.is <- format_SQL_in_statement(res$mukey)

# composite SQL WHERE clause
sql <- sprintf("mukey IN %s", mu.is)
```

```
# get commonly used map unit / component / chorizon records
# as a SoilProfileCollection object
# confusing but essential: request that results contain `mukey`
# with `duplicates = TRUE`
x <- fetchSDA(sql, duplicates = TRUE)

# safely set texture class factor levels
# by making a copy of this column
# this will save in lieu of textures in the original
# `texture` column
horizons(x)$texture.class <- factor(x$texture, levels = SoilTextureLevels())

# graphical depiction of the result
plotSPC(x, color='texture.class', label='compname',
        name='hzname', cex.names = 1, width=0.25,
        plot.depth.axis=FALSE, hz.depths=TRUE,
        name.style='center-center'
)



## query mukey + geometry that intersect with a bounding box

# define a bounding box: xmin, xmax, ymin, ymax
#
#         +-------------------(ymax, xmax)
#         |                         |
#         |                         |
#     (ymin, xmin) -----------------+
b <- c(-119.747629, -119.67935, 36.912019, 36.944987)

# convert bounding box to WKT
bbox.sp <-as(extent(b), 'SpatialPolygons')
proj4string(bbox.sp) <- '+proj=longlat +datum=WGS84'

# results contain associated map unit keys (mukey)
# return SSURGO polygons, after intersection with provided BBOX
ssurgo.geom <- SDA_spatialQuery(
  bbox.sp,
  what = 'geom',
  db = 'SSURGO',
  geomIntersection = TRUE
)

# return STATSGO polygons, after intersection with provided BBOX
statsgo.geom <- SDA_spatialQuery(
  bbox.sp,
  what = 'geom',
  db = 'STATSGO',
  geomIntersection = TRUE
)

# inspect results
```

```
par(mar = c(0,0,3,1))
plot(ssurgo.geom, border = 'royalblue')
plot(statsgo.geom, lwd = 2, border = 'firebrick', add = TRUE)
plot(bbox.sp, lwd = 3, add = TRUE)
legend(
  x = 'topright',
  legend = c('BBOX', 'STATSGO', 'SSURGO'),
  lwd = c(3, 2, 1),
  col = c('black', 'firebrick', 'royalblue'),
)


# quick reminder that STATSGO map units often contain many components
# format an SQL IN statement using the first STATSGO mukey
mu.is <- format_SQL_in_statement(statsgo.geom$mukey[1])

# composite SQL WHERE clause
sql <- sprintf("mukey IN %s", mu.is)

# get commonly used map unit / component / chorizon records
# as a SoilProfileCollection object
x <- fetchSDA(sql)

# tighter figure margins
par(mar = c(0,0,3,1))


# organize component sketches by national map unit symbol
# color horizons via awc
# adjust legend title
# add alternate label (vertical text) containing component percent
# move horizon names into the profile sketches
# make profiles wider
groupedProfilePlot(
  x,
  groups = 'nationalmusym',
  label = 'compname',
  color = 'awc_r',
  col.label = 'Available Water Holding Capacity (cm / cm)',
  alt.label = 'comppct_r',
  name.style = 'center-center',
  width = 0.3
)


mtext(
  'STATSGO (1:250,000) map units contain a lot of components!',
  side = 1,
  adj = 0,
  line = -1.5,
  at = 0.25,
  font = 4
)
```

```
}
```

`seriesExtent` *Retrieve Soil Series Extent Maps from SoilWeb*

Description

This function downloads a generalized representations of a soil series extent from SoilWeb, derived from the current SSURGO snapshot. Data can be returned as vector outlines (`SpatialPolygonsDataFrame` object) or gridded representation of area proportion falling within 800m cells (`raster` object). Gridded series extent data are only available in CONUS. Vector representations are returned with a GCS/WGS84 coordinate reference system and raster representations are returned with an Albers Equal Area / NAD83 coordinate reference system.

Usage

```
seriesExtent(s, type = c("vector", "raster"), timeout = 60)
```

Arguments

`s`	a soil series name, case-insensitive
`type`	series extent representation, `vector` results in a `SpatialPolygonsDataFrame` object and `raster` results in a `raster` object
`timeout`	time that we are willing to wait for a response, in seconds

Note

This function requires the `rgdal` package. Warning messages about the proj4 CRS specification may be printed depending on your version of `rgdal`. This should be resolved soon.

Author(s)

D.E. Beaudette

References

https://casoilresource.lawr.ucdavis.edu/see/

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

  # required packages
```

```
  library(sp)
  library(raster)
  library(rgdal)

  # specify a soil series name
  s <- 'magnor'

  # return as SpatialPolygonsDataFrame
  x <- seriesExtent(s, type = 'vector')
  # return as raster
  y <- seriesExtent(s, type = 'raster')

  # note that CRS are different
  proj4string(x)
  projection(y)

  # transform vector representation to CRS of raster
  x <- spTransform(x, CRS(projection(y)))

  # graphical comparison
  par(mar = c(1, 1 , 1, 3))
  plot(y, axes = FALSE)
  plot(x, add = TRUE)


}
```

`siblings` *Lookup siblings and cousins for a given soil series.*

Description

Lookup siblings and cousins for a given soil series, from the current fiscal year SSURGO snapshot via SoilWeb.

Usage

```
siblings(s, only.major=FALSE, component.data = FALSE, cousins = FALSE)
```

Arguments

`s`	character vector, the name of a single soil series, case-insensitive.
`only.major`	logical, should only return siblings that are major components
`component.data`	logical, should component data for siblings (and optionally cousins) be returned?
`cousins`	logical, should siblings-of-siblings (cousins) be returned?

Details

The siblings of any given soil series are defined as those soil series (major and minor component) that share a parent map unit with the named series (as a major component). Cousins are siblings of siblings. Data are sourced from SoilWeb which maintains a copy of the current SSURGO snapshot.

Value

sib `data.frame` containing siblings, major component flag, and number of co-occurrences

sib.data `data.frame` containing sibling component data

cousins `data.frame` containing cousins, major component flag, and number of co-occurrences

cousin.data `data.frame` containing cousin component data

Author(s)

D.E. Beaudette

References

soilDB Soil Series Query Functionality

Related tutorial.

See Also

OSDquery, siblings, fetchOSD

Examples

```
if(requireNamespace("curl") &
    curl::has_internet()) {

    # basic usage
    x <- siblings('zook')
    x$sib

    # restrict to siblings that are major components
    # e.g. the most likely siblings
    x <- siblings('zook', only.major = TRUE)
    x$sib
}
```

`simplifyColorData` *Simplify Color Data by ID*

Description

Simplify multiple Munsell color observations associated with each horizon.

Usage

```
simplifyColorData(d, id.var = "phiid", wt = "colorpct", bt = FALSE)
mix_and_clean_colors(x, wt='pct', backTransform = FALSE)
```

Arguments

`d`	a `data.frame` object, typically returned from NASIS, see details
`id.var`	character vector with the name of the column containing an ID that is unique among all horizons in `d`
`x`	a `data.frame` object containing sRGB coordinates associated with a group of colors to mix
`wt`	a character vector with the name of the column containing color weights for mixing
`bt`	logical, should the mixed sRGB representation of soil color be transformed to closest Munsell chips? This is performed by `aqp::rgb2Munsell`
`backTransform`	logical, should the mixed sRGB representation of soil color be transformed to closest Munsell chips? This is performed by `aqp::rgb2Munsell`

Details

This function is mainly intended for the processing of NASIS pedon/horizon data which may or may not contain multiple colors per horizon/moisture status combination. `simplifyColorData` will "mix" multiple colors associated with horizons in `d`, according to IDs specified by `id.var`, using "weights" (area percentages) specified by the `wt` argument to `mix_and_clean_colors`.

Note that this function doesn't actually simulate the mixture of pigments on a surface, rather, "mixing" is approximated via weighted average in the CIELAB colorspace.

The `simplifyColorData` function can be applied to data sources other than NASIS by careful use of the `id.var` and `wt` arguments. However, `d` must contain Munsell colors split into columns named "colorhue", "colorvalue", and "colorchroma". In addition, the moisture state ("Dry" or "Moist") must be specified in a column named "colormoiststate".

The `mix_and_clean_colors` function can be applied to arbitrary data sources as long as `x` contains sRGB coordinates in columns named "r", "g", and "b". This function should be applied to chunks of rows within which color mixtures make sense.

There are examples in the KSSL data tutorial and the soil color mixing tutorial.

Author(s)

D.E. Beaudette

simplifyFragmentData *Simplify Coarse Fraction Data*

Description

Simplify multiple coarse fraction (>2mm) records by horizon.

Usage

```
simplifyFragmentData(rf, id.var, nullFragsAreZero = TRUE)
```

Arguments

rf	a data.frame object, typically returned from NASIS, see details
id.var	character vector with the name of the column containing an ID that is unique among all horizons in rf
nullFragsAreZero	should fragment volumes of NULL be interpreted as 0? (default: TRUE), see details

Details

This function is mainly intended for the processing of NASIS pedon/horizon data which contains multiple coarse fragment descriptions per horizon. simplifyFragmentData will "sieve out" coarse fragments into the USDA classes, split into hard and para- fragments.

The simplifyFragmentData function can be applied to data sources other than NASIS by careful use of the id.var argument. However, rf must contain coarse fragment volumes in the column "fragvol", fragment size (mm) in columns "fragsize_l", "fragsize_r", "fragsize_h", and fragment cementation class in "fraghard".

There are examples in the KSSL data tutorial.

Author(s)

D.E. Beaudette

SoilWeb_spatial_query *Get SSURGO Data via Spatial Query*

Description

Get SSURGO Data via Spatial Query to SoilWeb

Usage

```
SoilWeb_spatial_query(bbox = NULL, coords = NULL, what = "mapunit", source = "soilweb")
```

Arguments

bbox	a bounding box in WGS84 geographic coordinates, see examples
coords	a coordinate pair in WGS84 geographic coordinates, see examples
what	data to query, currently ignored
source	the data source, currently ignored

Details

Data are currently available from SoilWeb. These data are a snapshot of the "official" data. The snapshot date is encoded in the "soilweb_last_update" column in the function return value. Planned updates to this function will include a switch to determine the data source: "official" data via USDA-NRCS servers, or a "snapshot" via SoilWeb.

Value

The data returned from this function will depend on the query style. See examples below.

Note

This function should be considered experimental; arguments, results, and side-effects could change at any time. SDA now supports spatial queries, consider using SDA_query_features instead.

Author(s)

D.E. Beaudette

Examples

```
if(requireNamespace("curl") &
    curl::has_internet()) {

    # query by bbox
    SoilWeb_spatial_query(bbox=c(-122.05, 37, -122, 37.05))

    # query by coordinate pair
    SoilWeb_spatial_query(coords=c(-121, 38))
}
```

STRplot *Graphical Description of US Soil Taxonomy Soil Temperature Regimes*

Description

Graphical Description of US Soil Taxonomy Soil Temperature Regimes

Usage

```
STRplot(mast, msst, mwst, permafrost = FALSE, pt.cex = 2.75, leg.cex = 0.85)
```

Arguments

mast	single value or vector of mean annual soil temperature (deg C)
msst	single value or vector of mean summer soil temperature (deg C)
mwst	single value of mean winter soil temperature (deg C)
permafrost	logical: permafrost presence / absence
pt.cex	symbol size
leg.cex	legend size

Details

Related tutorial.

Author(s)

D.E. Beaudette

References

Soil Survey Staff. 2015. Illustrated guide to soil taxonomy. U.S. Department of Agriculture, Natural Resources Conservation Service, National Soil Survey Center, Lincoln, Nebraska.

See Also

estimateSTR

Examples

```
par(mar=c(4,1,0,1))
STRplot(mast = 0:25, msst = 10, mwst = 1)
```

taxaExtent *Retrieve Soil Taxonomy Membership Grids*

Description

This function downloads a generalized representation of the geographic extent of any single taxa from the top 4 tiers of Soil Taxonomy. Data are provided by SoilWeb, ultimately sourced from from the current SSURGO snapshot. Data are returned as `raster` objects representing area proportion falling within 800m cells. Data are only available in CONUS and returned using an Albers Equal Area / NAD83 coordinate reference system.

Usage

```
taxaExtent(
  x,
  level = c("order", "suborder", "greatgroup", "subgroup"),
  timeout = 60
)
```

Arguments

`x`	single taxa name, case-insensitive
`level`	the taxonomic level within the top 4 tiers of Soil Taxonomy, one of `c('order','suborder','greatgrou`
`timeout`	time that we are willing to wait for a response, in seconds

Value

a `raster` object

Note

This is a work in progress.

Author(s)

D.E. Beaudette

Examples

```
if(requireNamespace("curl") &
   curl::has_internet()) {

  library(raster)

  # try a couple of different examples
```

```
  # soil order
  taxa <- 'vertisols'
  x <- taxaExtent(taxa, level = 'order')
  a <- raster::aggregate(x, fact = 5)

  # suborder
  taxa <- 'ustalfs'
  x <- taxaExtent(taxa, level = 'suborder')
  a <- raster::aggregate(x, fact = 5)

  # greatgroup
  taxa <- 'haplohumults'
  x <- taxaExtent(taxa, level = 'greatgroup')
  a <- raster::aggregate(x, fact = 5)

  # subgroup
  taxa <- 'Typic Haploxerepts'
  x <- taxaExtent(taxa, level = 'subgroup')
  a <- raster::aggregate(x, fact = 5)

  # quick evaluation of the result
  if(requireNamespace("rasterVis") & requireNamespace('viridis')) {
    rasterVis::levelplot(a,
      margin = FALSE, scales = list(draw = FALSE),
      col.regions = viridis::viridis,
      main = names(a)
    )
  }

  # slippy map
  if(requireNamespace("mapview")) {
   mapview::mapview(a, col.regions = viridis::viridis, na.color = NA, use.layer.names = TRUE)
  }



}
```

---

uncode *Convert coded values returned from NASIS and SDA queries to factors*

Description

These functions convert the coded values returned from NASIS or SDA to factors (e.g. 1 = Alfisols) using the metadata tables from NASIS. For SDA the metadata is pulled from a static snapshot in the soilDB package (/data/metadata.rda).

Usage

```
uncode(df, invert = FALSE, db = "NASIS",
       droplevels = FALSE,
       stringsAsFactors = default.stringsAsFactors()
       )
code(df, ...)
```

Arguments

df	data.frame
invert	converts the code labels back to their coded values (FALSE)
db	label specifying the soil database the data is coming from, which indicates whether or not to query metadata from local NASIS database ("NASIS") or use soilDB-local snapshot ("LIMS" or "SDA")
droplevels	logical: indicating whether to drop unused levels in classifying factors. This is useful when a class has large number of unused classes, which can waste space in tables and figures.
stringsAsFactors	logical: should character vectors be converted to factors? The 'factory-fresh' default is TRUE, but this can be changed by setting options(stringsAsFactors = FALSE)
...	arguments passed on to uncode

Details

These functions convert the coded values returned from NASIS into their plain text representation. It duplicates the functionality of the CODELABEL function found in NASIS. This function is primarily intended to be used internally by other soilDB R functions, in order to minimizes the need to manually convert values.

The function works by iterating through the column names in a data frame and looking up whether they match any of the ColumnPhysicalNames found in the metadata domain tables. If matches are found then the columns coded values are converted to their corresponding factor levels. Therefore it is not advisable to reuse column names from NASIS unless the contents match the range of values and format found in NASIS. Otherwise uncode() will convert their values to NA.

When data is being imported from NASIS, the metadata tables are sourced directly from NASIS. When data is being imported from SDA or the NASIS Web Reports, the metadata is pulled from a static snapshot in the soilDB package.

Beware the default is to return the values as factors rather than strings. While strings are generally preferable, factors make plotting more convenient. Generally the factor level ordering returned by uncode() follows the naturally ordering of categories that would be expected (e.g. sand, silt, clay).

Value

A data frame with the results.

Author(s)

Stephen Roecker

Examples

```
if(requireNamespace("curl") &
    curl::has_internet() &
    require(aqp)) {
  # query component by nationalmusym
  comp <- fetchSDA(WHERE = "nationalmusym = '2vzcp'")
  s <- site(comp)

  # use SDA uncoding domain via db argument
  s <- uncode(s,  db="SDA")
  levels(s$taxorder)
}
```

---

us_ss_timeline *Timeline of US Published Soil Surveys*

Description

This dataset contains the years of each US Soil Survey was published.

Usage

```
data("us_ss_timeline")
```

Format

A data frame with 5209 observations on the following 5 variables.

`ssa` Soil Survey name, a character vector

`year` year of publication, a numeric vector

`pdf` does a pdf exists, a logical vector

`state` State abbreviation, a character vector

Details

This data was web scraped from the NRCS Soils Website. The scraping procedure and a example plot are included in the examples section below.

Source

https://www.nrcs.usda.gov/wps/portal/nrcs/soilsurvey/soils/survey/state/

`waterDayYear` *Compute Water Day and Year*

Description

Compute "water" day and year, based on the end of the typical or legal dry season. This is September 30 in California.

Usage

```
waterDayYear(d, end = "09-30")
```

Arguments

`d`	anything the can be safely converted to `POSIXlt`
`end`	"MM-DD" notation for end of water year

Details

This function doesn't know about leap-years. Probably worth checking.

Value

A `data.frame` object with the following

`wy`	the "water year"
`wd`	the "water day"

Author(s)

D.E. Beaudette

References

Ideas borrowed from: https://github.com/USGS-R/dataRetrieval/issues/246 and https://stackoverflow.com/questions/48123049/create-day-index-based-on-water-year

Examples

```
# try it
waterDayYear('2019-01-01')
```

Index