Package ‘soilDB’

January 28, 2020

Type Package

Title Soil Database Interface

Version 2.5

Date 2020-01-27

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

Maintainer Dylan Beaudette <dylan.beaudette@usda.gov>

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

License GPL (>= 3)

LazyLoad yes

Depends R (>= 3.0.0)

Imports aqp, grDevices, graphics, stats, utils, plyr, xml2, sp,
reshape2, raster, curl, lattice

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

Repository CRAN

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

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

RoxygenNote 7.0.2

NeedsCompilation no

Date/Publication 2020-01-28 20:40:02 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`

`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 / absense
`saturated`	logical vector of seasonal saturation
`permafrost`	logical vector of permafrost presence / absense

Details

Pending.

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)
```

`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 |'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

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

Usage

```
fetchKSSL(series=NULL, bbox=NULL, mlra=NULL, pedlabsampnum=NULL,
pedon_id=NULL, pedon_key=NULL, returnMorphologicData=FALSE, simplifyColors=FALSE)
```

Arguments

`series`	a single soil series name, case insensitive
`bbox`	a bounding box in WGS84 geographic coordinates e.g. `c(-120,37,-122,38)`
`mlra`	a single MLRA ID, e.g. "18" or "22A"
`pedlabsampnum`	a single KSSL pedon lab sample number
`pedon_id`	a single user pedon ID
`pedon_key`	a single KSSL internal pedon ID
`returnMorphologicData`	optionally request basic morphologic data, see details section
`simplifyColors`	simplify colors (from morphologic data) and join with horizon data

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. The snapshots were last updated September 2018 (KSSL / NASIS).

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: horizon colors, rock fragments, pores, and structure. 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.

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

Function arguments (`series`, `mlra`, etc.) are NOT vectorized: the first element of a vector will be used when supplied as a filter. See the fetchKSSL tutorial for ideas on how to iterate over a set of IDs.)

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

References

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

See Also

`fetchOSD`

Examples

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

    # 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
if(requireNamespace("sp")) {
  par(mar=c(0,0,0,0))
  sp::plot(s, name='hzn_desgn', max.depth=150)
}
##
## morphologic data
##

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


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

# extract SPC
pedons <- s$SPC

## simplify color data manually
s.colors <- simplifyColorData(s$morph$phcolor, id.var = 'labsampnum', wt='colorpct')

# merge color data into SPC
h <- horizons(pedons)
h <- join(h, s.colors, by='labsampnum', type='left', match='first')
horizons(pedons) <- h

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

## 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)


# simplify fragment data
s.frags <- simplifyFragmentData(s$morph$phfrags, id.var='labsampnum')

# merge fragment data into SPC
h <- horizons(pedons)
```

```
    h <- join(h, s.frags, by='labsampnum', type='left', match='first')
    horizons(pedons) <- h


    # check
    par(mar=c(0,0,3,0))
    plot(pedons, color='total_frags_pct', 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 S3 R object specified by the aqp R package, known as a soil profile collection object. It flattens NASIS's pedon and component tables, including their various child tables, into several more easily managable 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 faclitate 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 interpretated 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

Examples

```
# check required packages
if(require("aqp") & requireNamespace("RODBC")) {

  # test that NASIS db connection is set up
  # note that you must setup this connection ahead of time
  # see inst/doc/setup_ODBC_local_NASIS.pdf
  if(any(grepl(names(RODBC::odbcDataSources()), pattern="nasis_local"))) {

    ## 1. fetchNASIS(from='pedon') NASIS setup
    # query depends on some pedon data in your selected set

    f <- try(fetchNASIS(from = 'pedons'))
    # note: wrap in try() to capture error in case of empty selected set

    # plot only those profiles with densic contact
    if(!inherits(f,'try-error')) {

      # which pedons have densic.contact==TRUE
      idx <- which(f$densic.contact)

      # if there are any pedons with densic contacts, plot them
      if(length(idx))
        plot(f[idx, ], name='hzname')

    } else { message(f[1]) }

    ## 2. fetchNASIS(from='component') NASIS setup:
    # perform a DMU-* query against the national database

    fc <- try(fetchNASIS(from = 'components'))
    # note: wrap in try() to capture error in case of empty selected set

    ## 3. fetchNASIS(from='pedon_report') NASIS setup:
    # run the 11-IND NASIS report 'fetchNASIS' against the national database
    # the result will automatically be opened and saved as fetchNASIS.txt
    # in NASIS Temp folder

    # the fetchNASIS.txt fileis read by fetchNASIS(from = 'pedon_report')
    # alternate: run offline against national db and supply `url` argument
    try(f <- fetchNASIS(from = 'pedon_report'))
    # note: wrap in try() to capture error in case of empty selected set
  }
}
```

`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 fuction 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

Examples

```
# check required packages
if(require(aqp) & requireNamespace("RODBC")) {

  # test that NASIS db connection is set up
  # note that you must setup this connection ahead of time
  # see inst/doc/setup_ODBC_local_NASIS.pdf
  if(any(grepl(names(RODBC::odbcDataSources()), pattern="nasis_local"))) {

    # query depends on some lab data, queried against the national database
    f <- try(fetchNASISLabData())
    # note: wrap in try in case no lab data in selected set

    # plot only those profiles with densic contact
    if(!inherits(f,'try-error')) {

      # which pedons have densic.contact==TRUE
      idx <- which(f$densic.contact)

      # if there are any pedons with densic contacts, plot them
      if(length(idx))
        plot(f[idx, ], name='hzname')
    } else { message(f[1]) }
  }
}
```

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 dataframe 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 Data by Soil Series Name*

Description

This functions fetches a varity 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

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, derrived from the current SSURGO snapshot

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

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

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

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

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

climate experimental climate summaries from PRISM stack

metadata metadata associated with SoilWeb cached summaries

Further details pending.

Value

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

Note

SoilWeb maintains a snapshot of the Official Series Description data. Please use the link above for the live data.

Author(s)

D.E. Beaudette

References

USDA-NRCS OSD search tools: http://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 fuction 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`

Examples

```
if(require(aqp)) {
  # path to local PedonPC back-end DB
  dsn <- "S:/Service_Center/NRCS/pedon/pedon.accdb"

    if(file.exists(dsn)) {
      # get routinely used soil data SoilProfileCollection object
      f <- fetchPedonPC(dsn)

      # determine which profiles have densic contacts
      idx <- which(f$densic.contact)

      # plot only those profiles with densic contact
      if(length(idx))
        plot(f[idx, ], name='hzname')
    }
}
```

`fetchRaCA` *Fetch KSSL Data (EXPERIMENTAL)*

Description

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

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`	an RaCA site id (e.g. 'C1609C01')
`get.vnir`	boolean, 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

`http://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,
                       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 formated 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 formated as an SQL WHERE clause (e.g. `WHERE = "areasymbol = 'IN001'"`. All functions are SQL querys 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`. Therefore if data from just SSURGO is desired, set `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 interpretated as NULLs. Set `nullFragsAreZero = TRUE` in those cases where NULL rock fragment values should be interpreted as 0.

Value

A dataframe or soil profile collection 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 that facilitates making spatial queries to Soil Data Access (SDA) based on 'mukey' or 'nationalmusym'. A typical 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.

Usage

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

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

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"))
}
```

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 dataframe 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 dataframe 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

```
# 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

```
# 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 dataframe.

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

```
# 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

```
# 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 dataframe.

Note

NULL total rock fragment values are assumed to represent an _absense_ 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 dataframe.

Note

NULL total rock fragment values are assumed to represent an _absense_ 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 dataframe.

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 dataframe.

Note

This fuction 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_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 dataframe.

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`,

Examples

```
## Example: export / convert DMS coordinates from NASIS and save to DD import file

# load required libraries
if(require(aqp) &
    require(soilDB) &
    require(rgdal) &
    require(plyr)) {

# get site data from NASIS
s <- try(get_site_data_from_NASIS_db())

if(!inherits(s, 'try-error')) {
  # keep only those pedons with real coordinates
  good.idx <- which(!is.na(s$x))
  s <- s[good.idx, ]

  ## this is not universally appropriate!
  # assume missing is NAD83
  s$horizdatnm[is.na(s$horizdatnm)] <- 'NAD83'

  # check: OK
  table(s$horizdatnm, useNA='always')

  # convert to NAD83
  old.coords <- cbind(s$x, s$y)

    if(nrow(s)) {
      # add temp column for projection information, and fill with proj4 style info
      s$proj4 <- rep(NA, times=nrow(s))
      s$proj4 <- paste('+proj=longlat +datum=', s$horizdatnm, sep='')

      # iterate over pedons, and convert to WGS84
      new.coords <- ddply(s, 'siteiid',
        .progress='text', .fun=function(i) {
          coordinates(i) <- ~ x + y
          proj4string(i) <- CRS(i$proj4)
          i.t <- spTransform(i, CRS('+proj=longlat +datum=WGS84'))
          i.c <- as.matrix(coordinates(i.t))
          return(data.frame(x.new=i.c[, 1], y.new=i.c[, 2]))
        })

      # merge in new coordinates
      s <- join(s, new.coords)

      # any changes?
      summary(sqrt(apply((s[, c('x', 'y')] - s[, c('x.new', 'y.new')])^2, 1, sum)))

     # save to update file for use with "Import of Standard WGS84 Georeference" calculation
      # in NASIS note that this defines the coordinate source as "GPS", hence the last
      # column of '1's.
```

```
        std.coordinates.update.data <- unique(cbind(s[, c('siteiid', 'y.new', 'x.new')], 1))
         # save to file
         write.table(std.coordinates.update.data,
                    file='c:/data/sgeoref.txt', col.names=FALSE,
                    row.names=FALSE, sep='|')
       }
   }}
```

`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 dataframe.

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 datbase (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

```
# 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

```
# 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 dataframe 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 dataframe.

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 dataframe 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 dataframe.

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 dataframe.

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-tranformed values with units of cm

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

Value

A list with the following components:

VG_curve estimated water retention curve: paired estimates of water potential and water content

VG_inverse_function function for converting measured 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)
}
```

mapunit_geom_by_ll_bbox

Fetch Map Unit Geometry from SDA

Description

Fetch map unit geometry from the SDA website by WGS84 bounding box.

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 `http://sdmdataaccess.nrcs.usda.gov`. See examples for bounding box formatting.

Value

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

Note

It appears that SDA does not actually return the spatial intersecion 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

References

http://casoilresource.lawr.ucdavis.edu/

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)
  } 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 a programatic interface 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-delimeted 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 geographicaly 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)

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

`http://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 suplied arguments.*

Description

Parse contents of a web report, based on suplied 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 specifiying the table to rerturn, or, NULL for a list of tables

Details

Report argument names can be infered 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
```

`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 GH issue page: `https://github.com/ncss-tech/soilDB/issues/61`.

`SDA_query` *Soil Data Access Query*

Description

Submit a query to the Soil Data Acccess (SDA) website in SQL, get the results as a dataframe.

Usage

```
SDA_query(q)
makeChunks(ids, size=100)
format_SQL_in_statement(x)
```

Arguments

`q`	a valid T-SQL query surrounded by double quotes
`ids`	vector of IDs for chunking, contents aren't used just length
`size`	target chunk size
`x`	character vector to be packed into an SQL 'IN' statement

Details

The SDA website can be found at `http://sdmdataaccess.nrcs.usda.gov` and query examples can be found at `http://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 dataframe containing the results. NULL is retutned when queries result in 0 matches rows.

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\n
    INNER JOIN mapunit mu ON mu.lkey = legend.lkey\n
    INNER JOIN component co ON mu.mukey = co.mukey\n
    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 \n
    component.mukey, cokey, comppct_r, compname, taxclname, \n
    taxorder, taxsuborder, taxgrtgroup, taxsubgrp \n
    FROM legend \n
    INNER JOIN mapunit ON mapunit.lkey = legend.lkey \n
    LEFT OUTER JOIN component ON component.mukey = mapunit.mukey \n
    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 \n
      WHERE mukey IN (SELECT DISTINCT mukey FROM\n
      SDA_Get_Mukey_from_intersection_with_WktWgs84('"
      , p, "')) ORDER BY mukey, cokey, comppct_r DESC")

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

`SDA_query_features` *Soil Data Access Spatial Query*

Description

Iterate over Spatial* object features and submit spatial queries to the SDA web-service.

Usage

```
SDA_query_features(x, id='pedon_id')
processSDA_WKT(d, g='geom', p4s='+proj=longlat +datum=WGS84')
```

Arguments

`x`	a `Spatial*` object with more than 1 feature, any defined coordinate system
`id`	the column name in `x` that contains a unique ID for each feature
`d`	`data.frame` returned by `SDA_query`, containing WKT representation of geometry
`g`	name of column in `d` containing WKT geometry
`p4s`	PROJ4 CRS defs

Details

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

Value

A dataframe containing the results.

Note

This function requires the ‘httr‘, ‘jsonlite‘, ‘XML‘, and ‘rgeos‘ packages

Author(s)

D.E. Beaudette

`seriesExtent` *Get/Display Soil Series Extent*

Description

Get or display the spatial extent of a named soil series using the Series Extent Explorer.

Usage

```
seriesExtent(s, timeout=60)
```

Arguments

`s`	the soil series name
`timeout`	time that we are willing to wait for a response, in seconds

Details

Soil series extent data are downloaded from a static cache of GeoJSON files on SoilWeb servers. Cached data are typically updated annually.

Value

when calling `seriesExtent`, a `SpatialPolygonsDataFrame` object

Note

This function require the ‘rgdal‘ package.

Author(s)

D.E. Beaudette

References

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

Examples

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

    # fetch series extent for the 'Amador' soil series
    s <- seriesExtent('amador')

    # plot SpatialPolygonsDataFrame
    if(require(sp))
      plot(s)
}
```

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", ...)
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`
`...`	further arguments passed on to `mix_and_clean_colors()`, see details
`x`	a `data.frame` object containing sRGB cordinates associated with a group of colors to mix
`wt`	a character vector with the name of the column containing color weights for mixing
`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 "colormoistst".

The `mix_and_clean_colors` funcion 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 / absense
`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)
```

`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 abbrevation, 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/

Examples

```
if (requireNamespace("curl") &
    curl::has_internet() &
    require("XML") &
    require("RCurl") &
    require("ggplot2") &
    require("gridExtra")
) {

data(state)
st <- paste0(c(state.abb, "PR", "DC", "VI", "PB"))
us_ss_timeline <- {
  lapply(st, function(x) {
    cat("getting", x, "\n")
    url <- getURL(paste0(
    "https://www.nrcs.usda.gov/wps/portal/nrcs/surveylist/soils/survey/state/?stateId=", x)
    )
    df  <- readHTMLTable(url, which = 22, stringsAsFactors = FALSE)
    df$state <- x
    return(df)
  }) ->.;
  do.call("rbind", .) ->.;
  names(.) <- c("ssa", "year", "pdf", "wss", "state")
  .[!grepl(.$year, pattern="current"), ] ->.;
}
us_ss_timeline <- within(us_ss_timeline, {
  ssa  = sapply(ssa, function(x) strsplit(x, "\r")[[1]][1])
  year = as.numeric(substr(year, 3,6))
  pdf  = ifelse(pdf == "Yes", TRUE, FALSE)
  wss  = NULL
})

test <- as.data.frame(table(us_ss_timeline$year), stringsAsFactors = FALSE)

g1 <- ggplot(data = test, aes(x = Var1, y = Freq)) +
  geom_histogram(stat = "identity") +
  xlab("Year") +
  ylab("Count") +
  theme(aspect.ratio = 1) +
  ggtitle("Number of Published \n US Soil Surveys by Year")
g2 <- ggplot(test, aes(x = Var1, y = cumsum(Freq))) +
  geom_histogram(stat = "identity") +
  xlab("Year") +
  ylab("Count") +
  theme(aspect.ratio = 1) +
  ggtitle("Cumulative Number of Published \n US Soil Surveys by Year")

grid.arrange(g1, g2, ncol = 2)

}
```

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 `PPOSIXlt`
`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