Package 'smwrBase'

November 24, 2015

Version 1.1.1

Date 2015-11-24

Title Functions to import and manipulate hydrologic data

Author Dave Lorenz <lorenz@usgs.gov>

Maintainer Laura DeCicco <ldecicco@usgs.gov>

Depends lubridate (>= 1.3)

Imports methods, stats, digest, memoise

Suggests smwrData (>= 0.6), dataRetrieval (>= 2.0.1), testthat

Description This package has data import and export functions for specialized formats used within the U.S. Geological Survey. It also contains several functions that are useful for managing or manipulating hydrologic and other data.

License CC0

LazyLoad yes

Collate 'timeDay-class.R' 'Arith-timeDay.R' 'LogPearsonIII.R' 'Math-timeDay.R' 'PearsonIII.R' 'anomalies.R' 'as.character.timeDay.R' 'as.data.frame.timeDay.R' 'as.timeDay.R' 'baseDay.R' 'baseDay2decimal.R' 'boxCox.R' 'c.timeDay.R' 'coalesce.R' 'conc.meq.R' 'conc2meq.R' 'daysInMonth.R' 'dectime.R' 'dectime2Date.R' 'dms2dd.R' 'eventProcessing.R' 'eventSeries.R' 'exportCSV.R' 'exportRDB.R' 'fillMissing.R' 'format.timeDay.R' 'fourier.R' 'group2row.R' 'hyperbolic.R' 'hysteresis.R' 'importCSV.R' 'importRDB.R' 'insertMissing.R' 'is.na.timeDay.R' 'isLike.R' 'length.timeDay.R' 'makeMeta.R' 'makepredictcall.R' 'mergeNearest.R' 'mergeQ.R' 'more.R' 'movingAve.R' 'movingDiff.R' 'na2miss.R' 'peaks.R' 'pick.R' 'print.timeDay.R' 'quadratic.R' 'readList.R' 'recode.R' 'regularSeries.R' 'sCurve.R' 'scaleRng.R' 'screenData.R' 'seasons.R' 'seqCollapse.R' 'setFileType.R' 'setTZ.R' 'shiftData.R' 'show-methods.R' 'smwrBase-package.R' 'sumComposition.R' 'untable.R' 'waterYear.R' 'whichRowCol.R' 'z%cn%.R' 'z[.timeDay.R'

NeedsCompilation no

R topics documented:

`smwrBase-package` *Data import, export and manipulation functions*

Description

This package has specialized functions for importing, managing, or manipulating hydrologic data.

Details

Package:	smwrBase
Type:	Package
Version:	1.1.1
Date:	2015-11-24
License:	File CC0
Depends:	methods,memoise,digest,lubridate

This package contains functions that import, manage, or manipulate hydrologic data and functions that apply specialized transformations used in hydrologic analyses amd modeling. A listing of the functions and their description is in the following table.

Function	Description
%cn%	Identify character strings that contain the specified pattern.
anomalies	Break down time-series data into long- and short-term deviations (anomalies) and the high-frequency variation.
as.timeDay	Convert data to objects of class "timeDay."
baseDay	Computes the "base" day of the year, a reference value that can be used to group days for the computation of summary statistics.
boxCox	Apply a Box-Cox power transformation.
coalesce	Merge a matrix or list of vectors selecting the first non-missing value.
conc.meq	A list containing necessary information for the function `conc2meq`.

conc2meq	Convert concentration in milligrams per liter to milli-equivalents per liter.
daysInMonth	The number of days in a month.
dectime	Convert dates and times to decimal time in years.
dectime2Date	Convert decimal time in years to an object of class "Date."
dlpearsonIII	The density of the log-Pearson Type III distribution.
dms2dd	Convert data in degrees, minutes, and seconds to decimal degrees.
dpearsonIII	The density of the Pearson Type III distribution.
eventLen	Compute the length or duration of an event.
eventNum	Compute the number of an event, identified by a TRUE value in a sequence.
eventSeq	Compute the sequence number within an event.
eventSeries	Create regular time-series data from recorded events.
exportCSV	Export a data frame to a comma-separated values file.
exportRDB	Export data to an ASCII relational-database file.
fillMissing	Interpolate missing values in a regular time-series of data.
fourier	Compute the Fourier series decomposition from date data.
group2row	Unstack data oriented in columns to rows of data.
hyperbolic	Apply a hyperbolic transformation.
hysteresis	Compute a basis for estimating hysteresis effect in some variable related to the argument x.
IboxCox	Apply the inverse Box-Cox power transformation.
Ihyperbolic	Apply the inverse hyperbolic transformation.
importCSV	Import a data frame from a comma-separated values file.
importRDB	Import a data frame from an ASCII relational-database file.
index.coalesce	Return the index column number instead of the values for the first non-missing value.
isCharLike	Determine whether the data be treated like character data.
IsCurve	Apply the inverse s-curve transformation.
isDateLike	Deterimne whether the data can be treated like date data.
isGroupLike	Determine whether the data can be treated like grouping data.
isNumberLike	Determine whether the data can be treated as numeric data.
makeMeta	Create a template meta file for a comma-separated values file.
mergeNearest	Merge two datasets by the nearest date and time.
mergeQ	Merge flow data with water-quality data.
miss2na	Convert a coded missing value to NA.
more	Display the contents of an object by pages.
movingAve	Compute the moving average in regular time-series data.
movingDiff	Compute the moving difference in regular time-series data.
na2miss	Convert NA to a coded missing value.
peaks	Compute the indices of peaks in time-series data.
pick	Select a value based on the value of a logical, integer, or character reference value.
plpearsonIII	Compute the cumulative probability of the log-Pearson Type III distribution.
ppearsonIII	Compute the cumulative probability of the Pearson Type III distribution.
qlpearsonIII	Compute the quantile of the log-Pearson Type III distribution.
qpearsonIII	Compute the quantile of the Pearson Type III distribution.
quadratic	Compute a basis for an orthogonal second-order polynomial.
readList	Import data arranged on lines into a list.
recode	Recode distinct values.
regularSeries	Put data collected at arbitrary times into a regular time series.
rlpearsonIII	Compute the random variates of the log-Pearson Type III distribution.
rpearsonIII	Compute the random variates of the Pearson Type III distribution.
sCurve	Apply the s-curve transformation.
scaleRng	Scale data to a specified range.
screenData	Screen data for missing values or gaps.

seasons	Create seasonal categories from dates.
seqCollapse	Collapse a sequence of integers to a compact character string.
setFileType	Support function to manage file suffixes.
setTZ	Set the time zone information for dates and times.
shiftData	Shift time-series data forward or backward.
sumComposition	Compute the percentages of data within a matrix.
timeDay	Various methods for manipulating time-of-day data, including conversion to and from character, addition, and others.
untable	Expand a 2-dimensional table into the raw values.
waterYear	Compute the water year of date data. The water year ends on September 30 of the year.
whichRowCol	Identify the row and column indexes for TRUE values in a logical matrix.

Author(s)

Dave Lorenz <lorenz@usgs.gov>

Maintainer: Dave Lorenz <lorenz@usgs.gov>

References

Lorenz, D.L., 2015, smwrBase—an R package for managing hydrologic data, version 1.1.1: U.S. Geological Survey Open-File Report 2015–1202, 7 p.

See Also

smwrData

`anomalies` *Anomalies*

Description

Decompose a series of observations into deviations (anomalies) from the mean for selected periods and the remainder (HFV or high frequency variation) using the method described in Appendix A of Vecchia (2000).

Usage

```
anomalies(x, ...)
```

Arguments

`x`	a time series (ts) or a vector of observations that represents a daily series. Missing values (NAs) are allowed only at the beginning and end of the series.
`...`	named anomalies and the length of the selected periods, generally in days. The anomalies must be specified in order of decreasing length.

Details

The intent of computing anomalies is to give flexibility in fitting the relation between flux, or concentration, and flow for time periods longer than a couple of years. Taking a very simple regression model:

$C = B0 + B1 * Q + e$,

where C is the concentration, $B0$ and $B1$ are the regression coefficients, Q is the flow, and e is the error. This can be re-expressed in terms of flow anomalies (for this example, 5- and 1-year anomalies are used, many others are possible):

$C = B0 + B1 * Qbar + B1 * A5 + B1 * A1 + B1 * HFV + e$,

where C, $B0$, $B1$, and e are the same as the simple regression, and $Qbar$ is the mean flow, $A5$ is the 5-year anomaly, $A1$ is the 1-year anomaly, and HFV is the high-frequency variation. The simple regression model assumes that the regression coefficient ($B1$) is constant for all anomalies. Computing anomalies removes that constraint and is represented by this model:

$C = B0 + B1 * A5 + B2 * A1 + B3 * HFV + e$,

where C, $A5$, $A1$, HFV, and e are the same as the re-expressed model, and $B0$, $B1$, $B2$, and $B3$ are regression coefficients (numerically different from the simple coefficients). $Qbar$ is a constant and is not needed for the regression.

Anomalies are computed sequentially. First, the mean of `x` is computed and subtracted from the data. Then for each anomaly, the running mean of the specified period is computed (the anomaly) and is subtracted from the data. The remainder is the HFV. This procedure ensures that the sum of the anomalies plus the mean is equal to the original data.

Value

A matrix of the specified anomalies and HFV. The mean of `x` is included as an attribute.

Note

The output matrix contains missing values in the beginning, corresponding to the length of the longest anomaly.

A long time-frame anomaly that is often of interest, is the 5-year anomaly, which is 1,826 days.

References

Vecchia, A.V., 2000, Water-quality trend analysis and sampling design for the Souris River, Saskatchewan, North Dakota, and Manitoba: U.S. Geological Survey Water-Resources Investigations Report 00-4019, 77 p.

Examples

```
## Not run:
library(smwrData)
```

```
data(Q05078770)
anomalies(log(Q05078770$FLOW), A3mo=90)

## End(Not run)
```

`Arith-timeDay` *Arithmetic Operators for* `timeDay` *objects*

Description

Addition of time-of-day data to either "Date" or "POSIXt" classes. This is useful when dates and times are recorded in separate columns in a dataset.

Usage

```
## S4 method for signature 'timeDay,POSIXt'
Arith(e1, e2)

## S4 method for signature 'POSIXt,timeDay'
Arith(e1, e2)

## S4 method for signature 'timeDay,Date'
Arith(e1, e2)

## S4 method for signature 'Date,timeDay'
Arith(e1, e2)
```

Arguments

`e1,e2`	timeDay and POSIXt or Date objects. Missing values are permitted in either argument and result in a missing value in the output.

Value

A vector of class "POSIXct."

Examples

```
as.Date("2001-03-04") + as.timeDay("10:00")
## Not run:
library(smwrData)
data(QW05078470)
# Note that the result is reported in the local time zone!
QW05078470$DATES + as.timeDay(QW05078470$TIMES)

## End(Not run)
```

as.character.timeDay *Character Vector*

Description

Convert the time-of-day object to a character string.

Usage

```
## S3 method for class 'timeDay'
as.character(x, ...)
```

Arguments

x	the time-of-day object to be converted.
...	not used, required for other methods.

Value

The values in x converted to a character representation.

as.data.frame.timeDay *Data Frames*

Description

Convert an object to class "data.frame."

Usage

```
## S3 method for class 'timeDay'
as.data.frame(x, row.names = NULL, optional = FALSE, ...,
  nm = deparse(substitute(x)))
```

Arguments

x	the time-of-day object to be converted.
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names to syntactic names is optional.
nm	the column name to create for x.
...	not used, required for other methods.

Value

A data frame is created containing the data in x.

See Also

as.data.frame (in base package)

as.timeDay *Methods for Function* as.timeDay

Description

Valid conversions for function as.timeDay.

Usage

```
as.timeDay(time, format)

## S4 method for signature 'timeDay,missing'
as.timeDay(time, format)

## S4 method for signature 'numeric,missing'
as.timeDay(time, format)

## S4 method for signature 'character,character'
as.timeDay(time, format)

## S4 method for signature 'character,missing'
as.timeDay(time, format)
```

Arguments

time	the time of day in character format.
format	The format for converting the time of day. See strptime for possible format specifiers.

Details

Inconsistent formats for time will result in an error. Missing values or empty strings in time will result in missing values in the output.

Examples

```
as.timeDay("10:00")
as.timeDay("3 PM", format="%I %p")
```

baseDay *Base Day*

Description

Computes the base day of the year, a reference value that can be used to group days for the computation of summary statistics.

Usage

```
baseDay(x, numeric = TRUE, year = c("calendar", "water", "climate"))
```

Arguments

x	a vector of class POSIXt, Dates, or character that represents a date. Missing values are permitted.
numeric	a logical value; TRUE means return the numeric value of the day, FALSE means return a factor.
year	a character string indicating the basis of the factor levels. See **Details**.

Details

The base day is computed such that all dates have the same reference value regardless of whether the year is a leap year or not. If year is "calendar," then the factor levels or day number begin on January 1; if year is "water," then the factor levels or day number begin on October 1; and if year is "climate," then the factor levels or day number begin on April 1.

Value

An integer value representing the base day number if numeric is TRUE. Otherwise a factor with levels for every day of the year.

Examples

```
# The default numeric result
baseDay(c("2000-02-29", "2000-03-01", "2001-03-01"))
# The result as a factor
baseDay(c("2000-02-29", "2000-03-01", "2001-03-01"), numeric=FALSE)
```

---

baseDay2decimal *Base Day*

Description

Computes the decimal time representation of the base day of the year.

Usage

```
baseDay2decimal(x)
```

Arguments

x	a vector of baseDay values, character, or factors of the form month abbreviation and day number, generally created from baseDay. Missing values are permitted and result in missing values in the output. Unmatched values also result in missing values in the output.

Value

A numeric value representing the base day.

See Also

baseDay

Examples

```
# The baseDay ordered by calendar year
bd.tmp <- baseDay(c("2000-02-29", "2000-03-01", "2001-03-01"),
  numeric=FALSE)
baseDay2decimal(bd.tmp)
# ordered by water year, result should agree
bd.tmp <- baseDay(c("2000-02-29", "2000-03-01", "2001-03-01"),
  numeric=FALSE, year="water")
baseDay2decimal(bd.tmp)
```

---

boxCox *Box-Cox Power Transform*

Description

Functions for transforming and back-transforming data using the Box-Cox power transform, with options to preserve the measurement units.

Usage

```
boxCox(x, lambda = 1, GM, alpha = 0)

IboxCox(x, lambda = 1, GM, alpha = 0)
```

Arguments

x	a numeric vector to be transformed by boxCox or back-transformed by IboxCox. Must be strictly positive for the forward trasnformation—the argument alpha can be used to force positive values. Missing values are allowed and result in corresponding missing values in the output. See **Details**.
lambda	the power term in the Box-Cox transformation. The value of 1 is a linear transform, the value of 0 results in a natural log transform.
GM	the value to use for the geometric mean of x. If not supplied, then compute the geometric mean (boxCox) or extract from the attributes of x (IboxCox).
alpha	an offset value for x.

Details

If x contains missing values, then GM is computed after omitting the missing values and the output vector has a missing value wherever x has a missing value.

The function boxCox computes the forward transform and the function IboxCox computes the inverse [boxCox] transform, or back-transform.

Value

A numeric vector of the transformed or back-transformed values in x with an attribute "GM" of the geometric mean.

Note

The original power transform described by Box and Cox (1964) is adjusted by a power transform of the geometric mean to retain the correct dimensional units of the original data as described in section 13.2 by Draper and Smith (1998).

References

Box, G.E.P., and Cox, D.R., 1964, An analysis of transformations: Journal of the Royal Statistical Society, v. 26, Series B, p. 211–243.
Draper, N.R., and Smith, H., 1998, Applied regression analysis: New York, John Wiley and Sons, 706 p.

See Also

hyperbolic

Examples

```
X.test <- c(1,4,9,16,25,36,49)
boxCox(X.test)
boxCox(X.test, lambda=0)
IboxCox(boxCox(1:3, lambda=0), lambda=0) # verify the back-transform
## Should return
# [1] 1 2 3
# attr(,"GM")
# [1] 1.817121
```

---

c.timeDay *Concatenate Data*

Description

Combine time-of-day data into a single vector.

Usage

```
## S3 method for class 'timeDay'
c(..., recursive = FALSE)
```

Arguments

recursive	not used, required for other methods.
...	any number of objects that can be converted to class "timeDay." The first must be a time-of-day object.

Value

A single vector of class "timeDay."

Examples

```
c(as.timeDay("10:00"), as.timeDay("3 PM", format="%I %p"))
```

`coalesce` *Replace missing values*

Description

Construct a vector with as few missing values as possible from a selected sequence of vectors.

Usage

```
coalesce(mat, ...)

index.coalesce(mat, ...)
```

Arguments

`mat`	a vector or matrix.
`...`	additional vectors or matrices, must have the same number of rows as `mat`. The last argument can be a constant that would substitute for all remaining missing values.

Value

For `coalesce`, a vector in which each element is determined by selecting the first non-missing value in the order in which they are specified in the argument list. The first step is to construct a matrix from all arguments. The output is initially set to column 1; for any missing values in the column, the data from column 2 are used and so on until all columns have been searched or all missing values replaced.

For `index.coalesce`, an integer vector indicating which column from mat or from the vectors or constant specified for ... produced the result in coalesce.

Note

This function is most useful for creating a column in a dataset from related columns that represent different methods. For example, a single column of alkalinity may be desired when there are multiple columns of alkalinity determined by various methods.

Examples

```
coalesce(c(1,NA,NA,3), c(2,2,NA,2))
# should be: [1]  1  2 NA  3
coalesce(c(1,NA,NA,3), c(2,2,NA,2), 0)
# should be: [1] 1 2 0 3
```

conc.meq *Special Data*

Description

Support function to supply data for functions within smwrBase.

Usage

```
conc.meq()
```

Value

A list containing necessary information for the function conc2meq.

Note

The user may choose to make local copies of the data with the same name (conc.meq) to be able to change or add to the list.

See Also

conc2meq

conc2meq *Concentrations to Milliequivalents*

Description

Convert concentrations in milligrams per liter (mg/L) to milli-equivalents per liter.

Usage

```
conc2meq(conc, constituent)
```

Arguments

conc	a numeric vector containing the concentration data in milligrams per liter.
constituent	the name of the constituent. Must be one of "aluminum," "ammonium," "bicarbonate," "bromide," "calcium," "carbonate," "chloride," "fluoride," "iron," "magnesium," "manganese," "nitrate as n," "nitrite as n," "phosphorus as p," "potassium," "sodium," "sulfate," or "sulfide." There must be enough characters in the name to uniquely identify the constituent. The case of the input name is ignored.

Value

Vector containing the milli-equivalent values. Missing values (NAs) are returned if the constituent name is invalid.

Note

The user must verify that the units of concentration are milligrams per liter. Only those constituents that are typically reported in milligrams per liter (rather than micrograms per liter) are provided in this function. Aluminum, iron, and manganese (and possibly sulfide) are sometimes reported in micrograms per liter. Such values should be divided by 1000.0 before using this function.

The conversion for iron assumes that the dissolved iron is iron II.

The conversion for phosphorus assumes that most of the phosphorus is divalent. The actual conversion for phosphorus is very dependent on pH.

The conversion factors are taken from table 9 (page 56) of Hem (1985). The available conversion factors are stored in the list created by `conc.meq` in smwrBase.

References

Hem, J.D., 1985, Study and interpretation of the chemical characteristics of natural water: U.S. Geological Survey Water-Supply Paper 2254, 263 p.

Examples

```
conc2meq(c(1,2,3), "Nitrate")
# should be: [1] 0.07139 0.14278 0.21417
```

`daysInMonth` *Days in a Month*

Description

Computes the number of days in a month or the total number of days in the year to the end of the month.

Usage

```
daysInMonth(month, year, cum = FALSE)
```

Arguments

`month`	the month number, must range in value from 1 to 12. Missing values are permitted.
`year`	the calendar year, replicated in length to match `month`. Missing values are permitted.
`cum`	a logical value. If `TRUE`, then return the cumulative days during the calendar year at the end of each month. If `FALSE`, then return the number of days in the month.

Value

A vector matching `month` of the requested number of days. Missing values are returned wherever either `month` or `year` is missing.

Examples

```
# Check February on a leap year and regular year.
# Should return 29, 28
daysInMonth(c(2,2), c(2000, 2001))
```

`dectime` *Decimal Time*

Description

Convert date/time data to be expressed as year and fractional part of year. This can be useful for plotting or representing time in a regression model.

Usage

```
dectime(dates, times, time.format, date.format, Date.noon = TRUE,
  year.type = c("calendar", "water", "climate"))
```

Arguments

`dates`	a vector of a valid date object, or character representation of dates. Missing values are permitted and produce corresponding missing values in the output.
`times`	a character representation of times. Missing values are permitted and produce corresponding missing values in the output.
`time.format`	format to convert `times`. See **Details**.
`date.format`	format to convert `dates` is character.
`Date.noon`	logical, if `TRUE` and `dates` is class "Date," then set set the time to noon, otherwise no time adjustment is made. See **Details**.
`year.type`	a character string indicating the type of year to determine the offset, must be one of "calendar," "water," or "climate."

Details

The format for `times` must be one of "hm," "hms," or "ms." Note that this is actually a conversion function, see **See Also**. If `times` is missing, `dates` is class "Date," and `Date.noon` is `TRUE`, then set the time to 12:00, so that the decimal time represents the center of the day.

Value

A vector representation of the data in decimal format–year and decimal fraction.

See Also

`hm` (in lubridate package), `strptime` (in base package)

Examples

```
dectime("11/11/1918", date.format="%m/%d/%Y")
dectime(1988:1990)
```

dectime2Date *Date Conversion*

Description

Convert time data expressed as year and fractional part of year to class "Date."

Usage

```
dectime2Date(x, Date.noon = TRUE)
```

Arguments

x	the decimal date to convert.
Date.noon	correct from noon correction for dectime.

Value

A vector of class "Date" corresponding to each value in x.

Note

A small value, representing about 1 minute, is added to each value in x to prevent truncation errors in the conversion. This can cause some errors if the data were converted from date and time data.

See Also

dectime, as.Date

Examples

```
dectime("02/07/2013", date.format="%m/%d/%Y")
# Convert back the printed result:
dectime2Date(2013.103)
```

dlpearsonIII *Log-Pearson Type III distribution*

Description

Density, cumulative probability, quantiles, and random generation for the log-Pearson Type III distribution.

Usage

```
dlpearsonIII(x, meanlog = 0, sdlog = 1, skew = 0)

plpearsonIII(q, meanlog = 0, sdlog = 1, skew = 0)

qlpearsonIII(p, meanlog = 0, sdlog = 1, skew = 0)

rlpearsonIII(n, meanlog = 0, sdlog = 1, skew = 0)
```

Arguments

x,q	vector of quantiles.
meanlog	vector of means of the distribution of the log-transformed data.
sdlog	vector of standard deviation of the distribution of the log-transformed data.
skew	vector of skewness of the distribution of the log-transformed data.
p	vector of probabilities.
n	number of observations. If length(n) > 1, then the length is taken to be the number required.

Details

Elements of x, q, or p that are missing will result in missing values in the returned data.

Value

Either the density (dlpearsonIII), cumulative probability (plpearsonIII), quantile (qlpearsonIII), or random sample (rlpearsonIII) for the described distribution.

Note

The log-Pearson Type III distribution is used extensively in flood-frequency analysis in the United States.

See Also

dpearsonIII, dlnorm (in stats package)

Examples

```
## Simple examples
dlpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qlnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
dlpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
## Simple examples
plpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qlnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
plpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
## Simple examples
qlpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qlnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
qlpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
## Simple examples
rlpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qlnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
rlpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
```

dms2dd *Decimal Degrees*

Description

Convert data in degrees, minutes, and seconds format to decimal degrees.

Usage

```
dms2dd(x, minutes = NULL, seconds = 0, split = "")
```

Arguments

x	a character or numeric vector coded as degrees, minutes, and seconds or a numeric vector of degrees. The character string may contain a leading zero for values less than 100. Missing values are permitted and result in missing values in the output.
minutes	a vector of minutes. If supplied, then x is assumed to be a numeric vector of degrees. Missing values are permitted.
seconds	a vector of seconds. Assumed to be 0 if not supplied. Missing values are permitted.
split	the delimiter for x if x is character.

Value

A numeric vector of decimal degrees the same length as x. Missing values are returned wherever x, minutes, or seconds has a missing value.

Examples

```
dms2dd(983206) # using a numeric value
# should be [1] 98.535
dms2dd("0983206") # using a character value
# should be [1] 98.535
dms2dd(98, 32, 6) # using numeric values for degrees, minutes and seconds
# should be [1] 98.535
dms2dd("98:32", split=":") # Note seconds not included in text
# should be [1] 98.53333
```

dpearsonIII *Pearson Type III distribution*

Description

Density, cumulative probability, quantiles, and random generation for the Pearson Type III distribution.

Usage

```
dpearsonIII(x, mean = 0, sd = 1, skew = 0)

ppearsonIII(q, mean = 0, sd = 1, skew = 0)

qpearsonIII(p, mean = 0, sd = 1, skew = 0)

rpearsonIII(n, mean = 0, sd = 1, skew = 0)
```

Arguments

x,q	vector of quantiles. Missing values are permitted and result in corresponding missing values in the output.
mean	vector of means of the distribution of the data.
sd	vector of standard deviation of the distribution of the data.
skew	vector of skewness of the distribution of the data.
p	vector of probabilities.
n	number of observations. If length(n) > 1, then the length is taken to be the number required.

Details

Elements of x, q, or p that are missing will result in missing values in the returned data.

Value

Either the density (dpearsonIII), cumulative probability (ppearsonIII), quantile (qpearsonIII), or random sample (rpearsonIII) for the described distribution.

Note

The log-Pearson Type III distribution is used extensively in flood-frequency analysis in the United States. The Pearson Type III forms the basis for that distribution.

See Also

dlpearsonIII, dnorm (in stats package)

Examples

```
## Simple examples
dpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
dpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
## Simple examples
ppearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
ppearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
## Simple examples
```

```
qpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
## compare to normal
qnorm(c(.5, .75, .9), 1.5, .25)
## Make a skewed distribution
qpearsonIII(c(.5, .75, .9), 1.5, .25, 0.25)
# Simple examples
rpearsonIII(c(.5, .75, .9), 1.5, .25, 0)
```

---

eventNum *Event Processing*

Description

Computes the event number eventNum, the length of events eventLen, or the sequence number for individual observations within an event eventSeq.

Usage

```
eventNum(event, reset = FALSE, na.fix = FALSE)

eventSeq(eventno)

eventLen(eventno, summary = FALSE)
```

Arguments

event	a logical vector where TRUE indicates that an event occurred. Missing values are treated as instructed by na.fix.
reset	a logical value indicating whether the event is assumed to continue until the next event, or only while event is TRUE.
na.fix	the value to use where event has missing values (NAs).
eventno	an integer vector indicating the event number. Generally the output from the eventNum function.
summary	a logical value, controlling output. See **Value** for details.

Value

The function eventNum returns an integer vector the same length as event indicating the event sequence number.

The function eventLen returns an integer vector the same length as eventno indicating the sequence length of the event if summary is FALSE, or a named integer vector indicating the sequence length of each event if summary is TRUE.

The function eventSeq returns an integer vector the same length as eventno indicating the sequence number of each element in the event.

Examples

```
## Notice the difference caused by setting reset to TRUE
eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE))
eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE), reset=TRUE)
## Notice the difference caused by setting reset to TRUE
eventSeq(eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE)))
eventSeq(eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE), reset=TRUE))
## Notice the difference caused by setting reset to TRUE
eventLen(eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE), reset=TRUE))
## This is an example of the summary option
eventLen(eventNum(c(TRUE,TRUE,FALSE,FALSE,TRUE,FALSE), reset=TRUE), summary=TRUE)
```

`eventSeries` *Regular Series*

Description

Some time-series analyses require data that are uniformly spaced in time. This function will construct a regular series from randomly spaced events.

Usage

```
eventSeries(times, period = "hour", which = "cumsum", begin, end,
  k.period = 1)
```

Arguments

`times`	a date-like vector corresponding to data.
`period`	character string that is valid input to the POSIXct method for the `seq` function is acceptable, specifying the spacing between successive periods. For example "year," "month," or "day."
`which`	a character string indicating the method to use. See **Details** for options.
`begin`	the beginning date as POSIXt or as character.
`end`	the end date as POSIXt or as character.
`k.period`	the number of units of `period` in each period of the output series.

Details

If there is no observation during a period, then that value is set to 0 if `which` is "sum" or the value for the previous period if `which` is "cumsum." The initial value of the series is always 0.

Value

The function `eventSeries` returns a data frame with two columns:

`DateTime`	the date and time.
`Sum`	the sum of the number of events in the period if `which` is "sum."
`CumSum`	the cumulative sum of the number of events up to and including the period if `which` is "cumsum."

Examples

```
## Not run:
library(smwrData)
data(QW05078470)
# Count the number of samples per month
with(QW05078470, eventSeries(DATES, "month", which="sum"))

## End(Not run)
```

---

exportCSV *Export Data*

Description

Exports a data frame to a text-based file.

Usage

```
exportCSV(x, file.name = "")
```

Arguments

x	the data frame to be written.
file.name	a character string naming the file for output.

Value

The file name is returned.

Note

The function exportCSV also writes a meta file that contains information about column formatting.

See Also

write.table (in utils package), importCSV

---

exportRDB *Export Data*

Description

Exports a data frame to a text-based file.

Usage

```
exportRDB(x, file.name = "data.rdb", col.names = NULL, meta = FALSE,
  code.rule = 10)
```

Arguments

x	the data frame to be written.
file.name	a character string naming the file for output.
col.names	a vector of column names to use instead of the column names in x.
meta	a logical value indicating whether the header should include a metadata template for documentation or not.
code.rule	an integer value indicating how many unique numeric values should be included in the metadata template for cases where each distinct value has a descriptive meaning rather than a numeric meaning.

Details

Setting the meta argument to TRUE generates a header that contains a template that can be edited by the user to describe the contents of the file.

Value

The file name is returned. write.table (in utils package), importRDB

---

fillMissing *Fill Missing Values*

Description

Replace missing values in time-series data by interpolation.

Usage

```
fillMissing(x, span = 10, Dates = NULL, max.fill = 10)
```

Arguments

x	the sequence of observations. Missing values are permitted and will be replaced.
span	the maximum number of observations on each side of each range of missing values to use in constructing the time-series model. See **Details**.
Dates	an optional vector of dates/times associated with each value in x. Useful if there are gaps in dates/times.
max.fill	the maximum gap to fill.

Details

Missing values at the beginning and end of x will not be replaced.

The argument span is used to help set the range of values used to construct the StructTS model. If span is set small, then the variance of epsilon dominates and the estimates are not smooth. If span is large, then the variance of level dominates and the estimates are linear interpolations. The variances of level and epsilon are components of the state-space model used to interpolate values, see StructTS for details. See **Note** for more information about the method.

If span is set larger than 99, then the entire time series is used to estimate all missing values. This approach may be useful if there are many periods of missing values. If span is set to any number less than 4, then simple linear interpolation will be used to replace missing values.

Value

The observations in x with missing values replaced by interpolation.

Note

The method used to interpolate missing values is based on tsSmooth constructed using StructTS on x with type set to "trend." The smoothing method basically uses the information (slope) from two values previous to missing values and the two values following missing values to smoothly interpolate values accounting for any change in slope. Beauchamp (1989) used time-series methods for synthesizing missing streamflow records. The group that is used to define the statistics that control the interpolation is very simply defined by span rather than the more in-depth measures described in Elshorbagy and others (2000).

If the data have gaps rather than missing values, then fillMissing will return a vector longer than x if Dates is given and the return data cannot be inserted into the original data frame. If Dates is not given, then the gap will be recognized and not be filled. The function insertMissing can be used to create a data frame with the complete sequence of dates.

References

Beauchamp, J.J., 1989, Comparison of regression and time-series methods for synthesizing missing streamflow records: Water Resources Bulletin, v. 25, no. 5, p. 961–975.

Elshorbagy, A.A., Panu, U.S., and Simonovic, S.P., 2000, Group-based estimation of missing hydrological data, I. Approach and general methodology: Hydrological Sciences Journal, v. 45, no. 6, p. 849–866.

See Also

tsSmooth (in stats package), StructTS (in stats package), insertMissing

Examples

```
## Not run:
library(smwrData)
data(Q05078470)
# Create missing values in flow, the first sequence is a peak and the second is a recession
Q05078470$FlowMiss <- Q05078470$FLOW
Q05078470$FlowMiss[c(109:111, 198:201)] <- NA
# Interpolate the missing values
Q05078470$FlowFill <- fillMissing(Q05078470$FlowMiss)
# How did we do (line is actual, points are filled values)?
par(mfrow=c(2,1), mar=c(5.1, 4.1, 1.1, 1.1))
with(Q05078470[100:120, ], plot(DATES, FLOW, type="l"))
with(Q05078470[109:111, ], points(DATES, FlowFill))
with(Q05078470[190:210, ], plot(DATES, FLOW, type="l"))
with(Q05078470[198:201, ], points(DATES, FlowFill))

## End(Not run)
```

format.timeDay *Encode in a Common Format*

Description

Format an object of class "timeDay."

Usage

```
## S3 method for class 'timeDay'
format(x, format, ...)
```

Arguments

x	the object to be formatted to type "character."
format	the format to use for output. See strptime for supported format information.
...	not used, required for other methods.

Value

A vector of character strings representing the time of day values in x.

See Also

strptime (in base package)

fourier *Fourier Series Components*

Description

Compute sine and cosine terms for describing annual or daily variations.

Usage

```
fourier(x, k.max = 1)
```

Arguments

x	a numeric vector where one unit specifies the period. See **Details**. Missing values are permitted.
k.max	the maximum number of paired sine and cosine terms specifying the order of the Fourier series.

Details

The argument x can be expressed as decimal time, either annual or diel; or it can be an object of class "Date," "POSIXct," or "POSIXlt" in which case it will be converted to annual decimal time using the dectime function.

Value

A matrix of the sine and cosine terms corresponding to the value—two terms are computed for each value of k from 1 to `k.max`: sine(k 2 pi x) and cosine(k 2 pi x). The value of `k.max` is included as an attribute.

Note

Water-quality data commonly follow a sinusoidal variation throughout a yearly cycle. A Fourier series of order one to three is generally enough to adequately describe that variation for many constituents.

See Also

`dectime`

Examples

```
# compute the sine and cosine terms for quarters of 2002
fourier(2002 + (0:3)/4)
#           sin(k=1)      cos(k=1)
# [1,]  3.54692e-014  1.00000e+000
# [2,]  1.00000e+000  7.08886e-013
# [3,] -3.65749e-013 -1.00000e+000
# [4,] -1.00000e+000 -3.78606e-013
# attr(, "k.max"):
# [1] 1
# Compare to 2 cycles per year:
fourier(2002 + (0:3)/4, 2)
#            sin(k=1)      cos(k=1)     sin(k=2) cos(k=2)
#[ 1,]  3.546924e-14  1.000000e+00 7.093848e-14        1
# [2,]  1.000000e+00  7.088855e-13 1.417771e-12       -1
# [3,] -3.657492e-13 -1.000000e+00 7.314983e-13        1
# [4,] -1.000000e+00 -3.786056e-13 7.572112e-13       -1
# attr(,"k.max")
# [1] 2
```

`group2row` *Restructure Data*

Description

Combine data from several rows into a single row based on common data in selected columns.

Usage

```
group2row(data, carryColumns, splitColumn, collectColumns)
```

Arguments

data	a data frame containing the columns to be combined.
carryColumns	the names of the columns that form a row in the output dataset. Each unique combination of values in these columns will be a new row in the output dataset.
splitColumn	the name of a single column. For each unique value in splitColumn and for each column in collectColumns, a new column is created in the output.
collectColumns	the names of the columns to be collected. See **Details**.

Details

The function group2row combines data from several rows into a single row. Certain columns in the input dataset are said to be "collected." Other columns may be "carried" into the output dataset by listing them in carryColumns. A new row will be created for each unique combination of values in the carryColumns. The output row consists of the carried columns plus new columns that are named by the unique values in the splitColumn concatenated with the names in the collectColumns. The number of columns in the output data frame is equal to the number of carryColumns plus the number of unique values in the splitColumn times the number of names in the collectColumns.

The strategy for collecting columns is to use a set of index values defined by the splitColumn. The maximum number of input rows collected for each output row is equal to the number of unique values defined in the splitColumn. The splitColumn is used to identify a column from the input data that contains output column information. If a row of input has a value in this column that matches one of the index values, then that row's data will be included in the output in the column positions corresponding to the matched index. The index values are concatenated with the input column names of the collected columns to derive output column names.

Examples

```
## Not run:
library(smwrData)
data(QWstacked)
group2row(QWstacked, c("site_no", "sample_dt", "sample_tm"), "parm_cd",
 c("result_va", "remark_cd"))

## End(Not run)
```

---

hyperbolic *Hyperbolic transform*

Description

Functions for transforming and back-transforming data using a hyperbolic function.

Usage

```
hyperbolic(x, factor = 0, scale = mean(x, na.rm = TRUE))

Ihyperbolic(x, factor = 0, scale)
```

Arguments

x	a numeric vector to be transformed by hyperbolic or back-trasnformed by Ihyperbolic. Must be strictly positive. Missing values are allowed. See **Details**.
factor	the hyperbolic adjustment term in the hyperbolic equation.
scale	the scaling factor for the data.

Details

If x contains missing values, then scale is computed after omitting the missing values and the output vector has a missing value wherever x has a missing value.

The basic equation for the hyberbolic transform is 1/(1 + (10^factor * x)/ scale). The basic equation is adjusted to produce fairly consistent values for small changes in factor and increase for increasing values in x.

The function hyperbolic computes the forward transform and the function Ihyperbolic computes the inverse [hyperbolic] transform, or back-transform.

Value

A numeric vector of the transformed or back-transformed values in x with an attribute "scale" of the values used for scale. The range of the values returned from hyperbolic is between 0 and 2 times scale.

Note

The original hyperbolic transform used a linear factor. The version in these functions uses the common log of the factor to make the factors easier to use.

When used with the default value for scale, factor values outside the range of +/- 3 have very little effect on the transform.

References

The use of a variable hyperbolic transform to help model the relations between stream water chemistry and flow was first described in:

Johnson, N.M., Likens, G.E., Borman, F.H., Fisher, D.W., and Pierce, R.S., 1969, A working model for the variation in stream water chemistry at the Hubbard Brook Experimental Forest, New Hampshire: Water Resources Research, v. 5, no. 6, p. 1353–1363.

See Also

boxCox

Examples

```
X.test <- c(1,4,9,16,25,36,49)
hyperbolic(X.test) # accept the defaults
hyperbolic(X.test, factor=1)
hyperbolic(X.test, factor=-1)
```

`hysteresis` *Compute Hysteresis*

Description

Compute a basis for estimating hysteresis effects in some variable related to the argument `x`.

Usage

```
hysteresis(x, step = 3)
```

Arguments

`x`	the sequence of observations. Missing values are permitted and will be copied in the output.
`step`	the number of previous observations to use to compute the local mean. See **Note**.

Value

A numeric vector that approximates the local trend in `x`.

Note

The basis for estimating hysteresis is the current value `x` minus the mean of the previous `step` values. The first `step` values in the output will be missing, and each missing value will result in `step` plus 1 missing values. This approximates the trend in `x`; if `x` is increasing in value over the previous `step` values, then the output will be positive and the greater the relative increase, the larger the output.

References

The use of hysteresis to help model the relations between stream water chemistry and flow are described in:

Garrett, J.D., 2012, Concentrations, loads, and yields of select constituents from major tributaries of the Mississippi and Missouri Rivers in Iowa, water years 2004-2008: U.S. Geological Survey Scientific Investigations Report 2012–5240, 61 p.

Wang, P., and Linker, L.C., 2008, Improvement of regression simulation in fluvial sediment loads: Journal of Hydraulic Engineering, v. 134, no. 10, p. 1,527–1,531.

See Also

`anomalies`

Examples

```
## Not run:
library(smwrData)
data(Q05078770)
# Plot flow and hysteresis to show looping
with(Q05078770, plot(log(FLOW), hysteresis(log(FLOW), 3), type="l"))

## End(Not run)
```

`importCSV` *Import Files*

Description

Imports a comma-separated variable file to a data frame.

Usage

```
importCSV(file.name = "", tz = "")
```

Arguments

`file.name`	a character string specifying the name of the comma-separated variable (CSV) file containing the data to be imported; `importCSV` requires `file.name` to be a readable file on the computer.
`tz`	a character string indicating the time-zone information for data imported as "POSIXct." The default is to use the local setting.

Details

All of the dates in a date column must have the same format as the first non-blank date in the column. Any date with a format different from that of the first non-blank date in the column will be imported as `NA` (missing value). Dates imported as class "Date" using a 4-digit year, 2-digit month, and 2-digit day with the period (.), hyphen (-), slash (/), or no separator. Time and date data are imported as class "POSIXct" and assumes the standard POSIX format for date and time.

Value

A data frame with one column for each data column in the CSV file.

Note

A NULL data frame is created if there are no data in the file.

See Also

`read.csv`, `read.table` (both in utils package), `scan`, `as.Date`, `as.POSIXct` (remainder in base package)

Examples

```
## Not run:
## These datasets are available in smwrData as text files
TestDir <- system.file("misc", package="smwrData")
TestPart <- importCSV(file.path(TestDir, "TestPart.csv"))

## End(Not run)
```

`importRDB` *Import Files*

Description

Imports a formatted, tab-delimited file to a data frame.

Usage

```
importRDB(file.name = "", date.format = NULL, tz = "",
  convert.type = TRUE)
```

Arguments

`file.name`	a character string specifying the name of the relational database (RDB) file containing the data to be imported.
`date.format`	a character string specifying the format of all date columns. Required for columns that contain date and time. The default value, `NULL`, will read any valid date (not date and time) format. The special formats "none," which suppresses date conversion; and "varies," which can be used when the date data included time data sometimes and sometimes not. For the latter special format, the date and time data must be in POSIX format (YYYY-mm-dd HH:MM) with optional seconds. For dates that are missing time data, the time will be set to midnight in the specified or local time zone.
`tz`	the time zone information of the data.
`convert.type`	logical `TRUE` or `FALSE`, convert data according to the format line? Setting `convert.type` to `FALSE` forces all data to be imported as character.

Details

All of the dates in a date column must have the same format as the first non-blank date in the column. Any date with a format different from that of the first non-blank date in the column will be imported as `NA` (missing value). By default, dates are imported as class "Date" using a 4-digit year, 2-digit month, and 2-digit day with the period (.), hyphen (-), slash (/), or no separator.

If a valid `date.format` is supplied, then the data are imported using `as.POSIXct`, and time information can be included in the data. If `date.format` is "none," then conversion of the date information is suppressed and the data are retained as character strings.

The value for `tz` should be a valid "Olson" format consisting typically of a continent and city. See `timezone` for a description of time zones. For the United States, use these time-zone specifications where daylight savings time is used:

Eastern	"America/New_York"
Central	"America/Chicago"
Mountain	"America/Denver"
Pacific	"America/Los_Angeles"
Alaska	"America/Anchorage"
Hawii	"America/Honolulu"

Use these time specifications where daylight savings time is not used: #'

Eastern	"America/Jamaica"
Central	"America/Managua"
Mountain	"America/Phoenix"
Pacific	"America/Metlakatla"

Value

A data frame with one column for each data column in the RDB file.

Note

A NULL data frame is created if there are no data in the file.

The header information contained in the RDB file is retained in the output dataset as `comment`.

If `convert.type` is `TRUE`, then non-numeric values, other than blanks, are converted to `NaN` (not a number) rather than `NA` (missing value) in numeric columns. `NaN` values are treated like `NA` values but can be identified using the `is.nan` function.

See Also

`read.table` (in utils package), `as.Date`, `as.POSIXct`, `comment` (remainder in base package)

Examples

```
## Not run:
## This dataset is available in smwrData as a text file
TestDir <- system.file("misc", package="smwrData")
TestFull <- importRDB(file.path(TestDir, "TestFull.rdb"))

## End(Not run)
```

---

insertMissing *Insert Missing Data*

Description

Inserts rows of missing values into a data frame for gaps in a sequence.

Usage

```
insertMissing(x, col, fill = FALSE)
```

Arguments

x	the data frame.
col	the name of the column that defines the sequence.
fill	logical (TRUE or FALSE), fill with many rows of data to match the sequence? If FALSE, insert only a single row. See **Note**.

Value

A data frame like x, but with rows of missing values where there is a break in the sequence in column col.

Note

Setting fill to TRUE is useful for setting up datasets for fillMissing if there are gaps in the retrieved data. Setting fill to FALSE, the default, is useful for creating a break in the sequence for plotting the data.

See Also

fillMissing, screenData

Examples

```
## Not run:
library(smwrData)
data(Q05078470)
# Plot the original data
with(Q05078470[100:120, ], plot(DATES, FLOW, type="l"))
# Remove 3 rows from the data set
Q05078470 <- Q05078470[-(109:111), ]
# Plot the data--line drawn through the missing record
with(Q05078470[100:117, ], lines(DATES, FLOW, col="green"))
# Insert a missing record
Q05078470 <- insertMissing(Q05078470, "DATES")
# Now plot to show gap in line
with(Q05078470[100:118, ], lines(DATES, FLOW, col="blue"))

## End(Not run)
```

---

is.na.timeDay *Missing Values*

Description

Indicate which elements are missing.

Usage

```
## S3 method for class 'timeDay'
is.na(x)
```

Arguments

x	the object to be tested.

Value

A logical vector of the same length as its argument x, containing TRUE for those elements marked NA and FALSE otherwise.

Examples

```
is.na(as.timeDay(c("10:30", "11:00")))
```

---

isCharLike *Test whether an object can be treated in a particular way*

Description

Tests if an object can be treated as a character, to name something; as a date; as a grouping variable, has distinct values; or as a number.

Usage

```
isCharLike(x)

isDateLike(x)

isGroupLike(x)

isNumberLike(x)
```

Arguments

x	any object.

Details

The function isCharLike tests whether x is of class "character" or "factor." The function isDateLike tests whether x is of class "Date" or "POSIXt." The function isGroupLike tests whether x is of class "character" or "factor" or if x is of type "integer" or "logical." The function isNumberLike tests whether x is of type "numeric" or of class "Date."

Value

A logical value TRUE if x meets the criteria, or FALSE if it does not.

Note

This function is most useful within other functions to control how that function handles a particular argument.

See Also

class, is.numeric, is.factor, is.character, is.integer, is.logical (all in base package)

Examples

```
## The first should be FALSE and the second TRUE
isCharLike(as.Date("2004-12-31"))
isCharLike("32")
## The first should be FALSE and the second TRUE
isDateLike(32)
isDateLike(as.Date("2004-12-31"))
## The first should be FALSE and the second TRUE
isGroupLike(as.Date("2004-12-31"))
isGroupLike(32)
## The first should be FALSE and the second TRUE
isNumberLike(as.Date("2004-12-31"))
isNumberLike(32)
```

---

length.timeDay *Length of an Object*

Description

Get the length of a time-of-day object.

Usage

```
## S3 method for class 'timeDay'
length(x)
```

Arguments

x	a time-of-day object.

Value

An integer of length 1 indicating the number of elements in x.

Examples

```
length(as.timeDay(c("10:30", "11:00")))
```

makeMeta *Metadata*

Description

Create a template meta file for a CSV file. The meta file is used by `importCSV` to define column types.

Usage

```
makeMeta(file.name = "")
```

Arguments

`file.name`	the name of the CSV file, which may include the path.

Value

The name of the meta file that was created.

Note

The meta file that is created will only contain column types of character, numeric, integer, and logical. It may need to be edited by the user to redefine the column types actually needed for the data, for example columns of class "Date," "POSIXct," or "factor."

See Also

`importCSV`

---

makepredictcall.quadratic *Utility Function for Safe Prediction*

Description

A utility to help `model.frame.default` create the correct matrices when predicting from models with `quadratic`, `hyperbolic`, or `boxCox` terms. Used only internally.

Usage

```
## S3 method for class 'quadratic'
makepredictcall(var, call)

## S3 method for class 'hyperbolic'
makepredictcall(var, call)

## S3 method for class 'boxCox'
makepredictcall(var, call)

## S3 method for class 'scaleRng'
makepredictcall(var, call)
```

Arguments

var	a variable.
call	the term in the formula, as a call.

Value

A replacement for call for the prediction variable.

Math-timeDay *Mathematical Functions for* timeDay *objects*

Description

No mathematical functions, such as log or exp are allowed on time-of-day data. An error results when trying to use any mathematical function on objects of class "timeDay."

Usage

```
## S4 method for signature 'timeDay'
Math(x)
```

Arguments

x	an object of class "timeDay."

---

mergeNearest *Merge Datasets*

Description

Merge two datasets based on the the nearest date between each observation.

Usage

```
mergeNearest(left, dates.left = "DATES", all.left = FALSE,
  suffix.left = "left", right, dates.right = "DATES",
  suffix.right = "right", Date.noon = TRUE, max.diff = "7 days")
```

Arguments

left	the left-hand dataset.
dates.left	the name of the column of dates in the left-hand dataset.
all.left	logical (TRUE or FALSE), include all rows from the left-hand dataset regardless of whether there is a matching row in the right-hand dataset?
suffix.left	the suffix to apply to common column names in the left-hand dataset.
right	the right-hand dataset.
dates.right	the name of the column of dates in the right-hand dataset.
suffix.right	the suffix to apply to common column names in the right-hand dataset.
Date.noon	logical (TRUE or FALSE), adjust columns of class "Date" to represent a noon observation rather than 12 a.m.?
max.diff	the maximum allowable difference in time for a match. See **Details**.

Details

The format for `max.diff` should be a numeric value followed by a description of the time span. The time span must be one of "secs," "mins," "hours," "days," or "weeks" for seconds, minutes, hours, days, or weeks, respectively.

Value

A data frame of the merged data with common column names renamed by the `suffix` arguments to avoid conflict.

Note

Water-quality data taken at a specific time frequently need to be merged with daily flow data or merged with other water-quality data such as replicate samples or samples of a different medium taken at about the same time, but having a different time stamp.

See Also

`mergeQ`

Examples

```
library(smwrData)
data(Q05078470)
data(QW05078470)
# Set the actual time of sampling in QW05078470
QW05078470 <- transform(QW05078470, DATES=DATES + as.timeDay(TIMES))
mergeNearest(QW05078470, right=Q05078470)
# Notice the difference in selected dates
mergeNearest(QW05078470, right=Q05078470, Date.noon=FALSE)
```

---

mergeQ *Merge Flow into another Dataset*

Description

Merges the flow (or other data) column from one or many daily-value datasets into a another dataset with one or more stations.

Usage

```
mergeQ(QWdata, STAID = "STAID", FLOW = "FLOW", DATES = "DATES",
  Qdata = NULL, Prefix = NULL, Plot = TRUE, ...)
```

Arguments

`QWdata`	a data frame with at least a date column on which to merge.
`STAID`	a character string of the name of the station-identifier column. The column name must agree in the `QWdata` and flow datasets.
`FLOW`	a character string of the name of the flow column. The column name must agree in flow datasets and will be the column name in the merged dataset. See **Details**

DATES	a character string of the name of the column containing the date information. The column name must agree in QWdata and flow datasets. All datasets must be sorted by date.
Qdata	a data frame containing daily-flow values.
Prefix	a character string indicating the prefix of the names of datasets containing daily-flow values.
Plot	a logical value indicating whether to plot the joint distribution of sampled flows and observed flows. See **Notes** for a description of the plot. Used only if a single column is specified in FLOW.
...	defines the dataset containing daily flow values for each station identifier.

Details

More than one column can be specified for FLOW when merging a single station, and the flow data are specified in Qdata.

Value

A data frame like QWdata with an attached flow column(s).

Note

The station-identifier columns must be of class character.

There are fours ways to merge flow and water-quality data:

A dataset that contains data for a single site does not require a STAID column. Qdata must be supplied. This case must be used if the flow record is incomplete or does not cover the range of dates in QWdata; all other methods will fail if that is the case. See Example 1.

A dataset that contains data for one or more sites can be merged with a dataset that contains flow data for the sites in that first dataset. This method will fail if there is not a complete list of station identifiers in the flow dataset. See Example 2.

A dataset that contains data for one or more sites can be merged with flow datasets that have names based on STAID. The structure of the name must be some common prefix followed by the station identifier. The station identifier must conform to a valid name. This method will fill in missing values (NAs) if a dataset corresponding to a station identifier is not available. See Example 3.

A dataset that contains data for one or more sites can be merged with flow datasets that have arbitrary names. Station identifiers that do not conform to valid names must be quoted. This method will fail if there is not a complete list of station identifiers supplied as arguments. See Example 4.

The plot shows the joint distribution of the sampled flows and observed flows from the sampling time period. The quantile-quantile plots are used to assess whether the sampled and observed flows have the same distribution. If the distributions are the same, then the plot will be approximately a straight line (included as a reference line). The extreme points can have more variability than points toward the center. A plot that shows the upper end trailing upward indicates that the largest flows have been under sampled and the sampled data may not give a reliable estimate of loads.

See Also

mergeNearest

Examples

```
## Not run:
library(smwrData)
data(Q05078470)
data(Q05078770)
data(Qall)
data(QW05078470)
data(QWall)
#                   Example 1
#
mergeQ(QW05078470, Qdata=Q05078470, Plot=FALSE)

#                   Example 2
#
mergeQ(QWall, FLOW="Flow", Qdata=Qall, Plot=FALSE)

#                   Example 3
#
mergeQ(QWall, Prefix="Q", Plot=FALSE)

#                   Example 4
# Note quotes required for station identifiers
mergeQ(QWall, "05078470"=Q05078470, "05078770"=Q05078770, Plot=FALSE)

## End(Not run)
```

---

more *Display Data*

Description

Display the contents of an object by pages.

Usage

```
more(x, n = 20L, ...)
```

Arguments

x	any valid object, generally a data frame, matrix, or table.
n	a positive integer indicating how many lines to print for a page.
...	additional arguments to be passed to methods for head or tail.

Value

The object x is returned invisibly. Sections of x of length n are displayed during the execution of the function.

Note

The function more is intended for interactive sessions. If used in a non-interactive session, it simply returns x invisibly.

Several keyboard commands can be used to view the contents of x. The function more will display n lines of x and wait for user input. Any of the following commands can be entered by the user; either upper- or lowercase letters are accepted.

q Quit

t Go to top of x

b Go to bottom of x

u Go up 1/2 page

p Go to previous page

d Go down 1/2 page

colName/pattern Search for pattern in the column named colName

h or ? Print help

any other letter Go down full page

Searching for a pattern in a column uses grep to search for the specified pattern in the character representation of the data in the column. This makes it possible to search columns that are not type character.

See Also

head and tail (both in utils package), grep (in base package)

---

movingAve *Moving Averages*

Description

Implements the Savitzky-Golay (Savitzky and Golay, 1964) filter on a regular series of data to compute a moving average.

Usage

```
movingAve(x, span = 3, order = 0, pos = "center")
```

Arguments

x	the data to be averaged or differenced. Missing values are permitted but result in missing values in the output.
span	the length of the data to be averaged.
order	the polynomial order for averaging. Must be less than span.
pos	how to position the output data relative to the value returned; "center" means that the value represents the average of the most central value relative to the span, "end" or "trailing" means the the value is the average of the preceding span values, and "begin" or "leading" means the value is the average of the following span values.

Value

A vector of the same length as x containing the averages.

Note

For odd values of span and pos equal to "center," order equal to 0 or 1 gives the same result.

In general, there is no reason to use polynomial orders greater than 2, and pos should always be set to "center" for polynomial orders greater than 1 to avoid strange behavior due to end effects.

The weights for the averages are computed based on linear model theory (Savitzky and Golay, 1964; Wood and Hockens, 1970). Wood and Hockens (1970) also discuss some artifacts resulting from smoothing.

References

Savitzky, A., and Golay, M.J.E., 1964, Smoothing and differentiation of data by simplified least squares procedures: Analytical Chemistry, v. 36, no. 8, p. 1627–1639.

Wood, L.C., and Hockens, S.N., 1970, Least squares smoothing operators: Geophysics, v. 35, no. 6, p. 1005–1019.

See Also

filter (in stats package), diff (in base package), movingDiff

Examples

```
## Construct a simple valley
movingData <- abs(seq(-5, 5))
movingAve(movingData, span=5)
movingAve(movingData, span=5, order=2)
```

---

movingDiff *Moving Differences*

Description

Filter a regular series of data to compute a moving difference.

Usage

```
movingDiff(x, span = 1, pos = "end")
```

Arguments

x	the data to be averaged or differenced. Missing values are permitted but result in missing values in the output.
span	the span of the differences.
pos	how to position the output data relative to the value returned; "center" means that value represents the difference between the preceding span/2 value and the following span/2 value, "end" or "trailing" means that value is the difference between the preceding span value and the current value, and "begin" or "leading" means that value is the difference between the current value and the following span value.

Value

A vector of the same length as x containing the differences.

See Also

filter (in stats package), diff (in base package), movingAve

Examples

```
# Construct a simple valley
movingData <- abs(seq(-5, 5))
movingDiff(movingData, span=1)
# Compare to diff:
diff(movingData)
```

---

na2miss *Recode Data*

Description

Converts missing values (NAs) to or from a user specified value.

Usage

```
na2miss(x, to = -99999)

miss2na(x, from = -99999)
```

Arguments

x	a vector. Missing values (NAs) are allowed.
to	the replacement value for NA.
from	the target value to match and replace with NA.

Value

An object like x with each target value replaced by the specified value.

Note

The function na2miss converts missing values (NA) to the value to and is useful to prepare a vector for export and subsequent use by software external to R that does not handle NAs.
The function miss2na converts the value from to NA and can be used to recode data imported from external software that uses a special value to indicate missing values.

See Also

is.na, sub (both in base package)

Examples

```
## Construct simple substitutions
na2miss(c(1, 2, 3, NA, 5, 6))
```

---

peaks *Find Local Maxima*

Description

Find the local maxima in a vector.

Usage

```
peaks(x, span = 3, ties = "first", ends = TRUE)
```

Arguments

x	any numeric vector. Missing values are permitted, but suppress identifying peaks within span.
span	The window width, the default value is 3, meaning compare each value to both of its neighbors. The value for span must be odd and if set to an even value, then it is increased to the next largest odd value.
ties	a character indicating how to handle ties. See **Details**.
ends	a logical value indicating whether or not to include either the first or last observations in the sequence if it is a local maximum.

Details

Possible values for ties include "none," which treats sequential tied values as individual values; all other values can be thought of as collapsing sequential tied values—"first," "middle," or "last" identify the first, middle, or last, respectively, of a sequence of ties as the peak if appropriate.

Value

A vector matching x of logical values indicating whether the corresponding element is a local maximum or not.

Note

A peak is defined as an element in a sequence that is strictly greater than all other elements within a window of width span centered at that element. As such, setting ties to "none" has the effect of not identifying peaks with sequential tied values.

See Also

max (in base package)

Examples

```
# Note the effect of missing values
peaks(c(1:6,5,4,NA,4,6,9,NA))
peaks(c(1:6,NA,5,4,NA,4,6,9,NA))
# Note the effect of ties
peaks(c(1:6,6,6,5,4,3,4,6,9))
peaks(c(1:6,6,6,5,4,3,4,6,9), ties="none")
```

---

pick *Conditional Element Selection*

Description

Return the value associated with test from the supplied vectors.

Usage

```
pick(test, ..., .pass = test, na = NA)
```

Arguments

test	a logical, numeric, or character vector that indicates which value to select from the data supplied in ... See **Details**.
.pass	the value to return for any element of test that does not match an argument name in ... Useful only when the class of test is "character" or "factor."
na	the value to return for any element of test is NA.
...	the values to be selected.

Details

If test is logical, then if test is TRUE, return the first argument in ..., otherwise return the second argument.
If test is numeric, then return that value in the list defined by ...
If test is character, then return that value in the list defined by ..., which must be named in the call.
If test is NA, then return the value specified by na.

Value

A vector of the same length as test and data values from the values list defined by ... The mode of the result will be coerced from the values list defined by ...

Note

This function is designed to replace nested `ifelse` expressions. See **Examples**. It is different from `switch` in that the value selected from the possible alternatives is selected by the values in `test` rather than by a single value.

See Also

`ifelse`, `switch` (both in base package)

Examples

```
## Create the test vector
testpick <- c(1,2,3,1)
## Nested ifelse
ifelse(testpick == 1, 1,
 ifelse(testpick == 2, 3,
   ifelse(testpick == 3, 5, NA)))
## Results by pick:
pick(testpick, 1, 3, 5)
## Create a test vector of character data
testpick <- c("a","b","c","a")
pick(testpick, a=1, b=3, c=5)
```

`print.timeDay` *Print an Object*

Description

Print an object of class "timeDay."

Usage

```
## S3 method for class 'timeDay'
print(x, ...)
```

Arguments

`x`	an object of class "timeDay."
`...`	not used, required for other methods.

Value

The object `x` is returned invisibly.

Side Effect

The object `x` is printed.

Note

The object is printed using the format that created the object in `as.timeDay`.

See Also

timeDay-class, as.timeDay

---

quadratic *Linear and Quadratic Terms*

Description

Computes orthogonal polynomials of degree 2 (Cohn and others, 1992). Used primarily in a linear regression formula.

Usage

```
quadratic(x, center = NULL)
```

Arguments

x	a numeric vector. Missing values are permitted and result in corresponding missing values in the output.
center	an optional value to use for the center of x.

Value

A matrix of two columns—the centered value of x and its square.

Note

If center is specified, then the polynomials will not necessarily be orthogonal. If used in a linear regression formula, then the coefficient of the linear term is the slope at center.
The function quadratic differs from poly in that the data are not scaled, so the regression coefficients are directly interpretable in terms of the units of x.

References

Cohn, T.A., Caulder, D.L., Gilroy, E.J., Zynjuk, L.D., and Summers, R.M., 1992, The validity of a simple statistical model for estimating fluvial constituent loads—An empirical study involving nutrient loads entering Chesapeake Bay: Water Resources Research, v. 28, no. 5, p. 937–942.

See Also

poly (in stats package)

Examples

```
## first and second orthogonal polynomials for the sequence from 1 to 10
quadratic(seq(10))
```

readList *Import Data*

Description

Import data arranged on lines into a list.

Usage

```
readList(file, names = TRUE, sep = "", nlines = 1, convert = NULL)
```

Arguments

file	a character string specifying the name of the file.
names	logical, if TRUE, then take component names from the first entry in the line. If FALSE, then the components are sequentially numbered.
sep	the separator character for the data in each line. If a blank string (the default), then any white space is taken as the separator.
nlines	the number of lines that represent a single collection of data,
convert	character string indicating how to convert the data. Must be a valid value for the Class argument of as.

Value

A list with one component for each nlines in the input file.

See Also

as (in methods package)

Examples

```
# Make a 3-line example dataset with component names A, B, and C.
cat("A 1 2 3 4\nB 5 6 7\nC 8 9\n", file="readList.test")
# Read the example dataset
readList("readList.test")
```

recode *Recode Data*

Description

Converts a specified value to another value.

Usage

```
recode(x, from, to)

## S3 method for class 'factor'
recode(x, from, to)

## S3 method for class 'integer'
recode(x, from, to)

## S3 method for class 'character'
recode(x, from, to)

## S3 method for class 'numeric'
recode(x, from, to)
```

Arguments

x	a vector. Missing values (NAs) are allowed.
from	the target value to match and replace.
to	the replacement value.

Value

An object like vector with each target value replaced by the specified value.

Note

When used on numeric (type "double"), the recode function uses an approximate match within a small tolerance range to avoid mismatches due to computations.
The function sub offers greater flexibility than recode for replacing parts of text instead of the complete text.

See Also

sub (in base package), na2miss, miss2na

Examples

```
XT <- c(1, 2, 0, 4)
recode(XT, 0, 3)
```

---

regularSeries *Regular Series*

Description

Some time-series analyses require data that are uniformly spaced in time. This function will construct a regular series from randomly spaced data using any of several user-definable methods.

Usage

```
regularSeries(x, times, period = "month", which = "middle", begin, end,
  k.period = 1)
```

Arguments

x	a vector of observations that represents a series.
times	a date-like vector corresponding to data.
period	character string that is valid input to the POSIXct method for the function seq is acceptable, specifying the spacing between successive periods. For example "year," "month," or "day."
which	a character string indicating the method to use, or the name of a function. See **Details** for options.
begin	the beginning date as POSIXt or as character.
end	the end date as POSIXt or as character.
k.period	the number of units of period in each period of the output series.

Details

For regularSeries, if there is no observation during a period, then that value is set to NA. If there is one observation, then the value is set to the value of that single observation. The value of which controls how periods with multiple observations are handled. Three character strings are recognized for selecting a single value: "earliest" selects the earliest observation in the period, "middle" selects the observation closest to the middle of the period, and "latest" selects the latest observation in the period. If which is not one of these, then it should be the name of a function such as mean or median.

Value

The function regularSeries returns a data frame with the following columns:

Season	the season number.
SeasonStartDate	the starting date of the corresponding season number—the season includes dates greater than or equal to this date.
SeasonEndDate	the end date of the corresponding season number—the season includes dates strictly less than this date.
Value	the value from x for the corresponding season number.
ValueDate	the date from times for the corresponding season number if which was one of "earliest," "middle," or "latest"; otherwise missing.

Examples

```
## Not run:
library(smwrData)
data(QW05078470)
with(QW05078470, regularSeries(P00665, DATES))
# there should be no values for season numbers 2, 5, or 10

## End(Not run)
```

scaleRng *Scale Data*

Description

Transforms numeric data to a specified range.

Usage

```
scaleRng(x, Min = 0, Max = 1, x.range = range(x, na.rm = TRUE))

IscaleRng(x, Min, Max, x.range)
```

Arguments

x	any numeric vector. Missing values are permitted and result in missing values in the corresponding output.
Min	the minimum of the output range.
Max	the maximum of the output range.
x.range	the input range to map to the output range. The default range is computed from the range of x after removing missing values.

Details

The function scaleRng maps the minimum of x.range to Min and the maximum of x.range to Max and uses linear interpolation for other values in x.

Value

A numeric vector scaled to the specified range.

Note

Some applications recommend or require data scaled to a consistent range. The function scaleRng will do that and can be used to back-transform the data.

Examples

```
## simple case with back-transform
x.tmp <- print(scaleRng(c(1.2, 2.3, 3.4, 5.6)))
IscaleRng(x.tmp)
## now set the expected ranges
x.tmp <- print(scaleRng(c(1.2, 2.3, 3.4, 5.6), x.range=c(1, 6)))
IscaleRng(x.tmp)
```

screenData *Screen Data for Completeness*

Description

Screens data to determine if a value is reported for each date by calendar or water year.

Usage

```
screenData(dates, values, type = "DV", year = "calendar", printit = TRUE)
```

Arguments

dates	the sequence of dates for each value in values.
values	the sequence of observations.
type	the frequency of values. Only daily values ("DV") and intermittent, or discrete, values ("IV") are accepted in this version. The whole text is required, but not case sensitive.
year	the type of year: "calendar" or "water," which begins on October 1 of the previous calendar year and ends on September 30.
printit	logical, if TRUE, then print the results in an

Details

Missing values are permitted in either dates or values. Those missing values are tallied in the completeness of record.

Value

For type = "DV," a matrix of the counts of *missing* values, either coded as NA or not in the dataset, for each month and each year within the range of dates.

For type = "IV," a matrix of the counts of *observed* values for each month and each year within the range of dates.

References

This function is based on the screen program described in:
Rutledge, A.T., 2007, Program user guide for RECESS: *at* http://water.usgs.gov/ogw/recess/UserManualRECESS.pdf.

Examples

```
library(smwrData)
data(Q05078770)
# this should indicate no missing values.
with(Q05078770, screenData(DATES, FLOW))
# There should be missing values shown for:
#months 10-12 in water year 2003 (October - December, 2002), and
#months 1-9 of water year 2004.
with(Q05078770, screenData(DATES, FLOW, year="w"))
```

sCurve *S-Curve Transform*

Description

Functions for transforming and back-transforming data using an s-shaped curve.

Usage

```
sCurve(x, location = 0, scale = 1, shape = 1)

IsCurve(x, location = 0, scale = 1, shape = 1)
```

Arguments

x	a numeric vector to be transformed by sCurve or back-transformed by IsCurve. Missing values are allowed and result in corresponding missing values in the output.
location	the transition point in the s-curve transform.
scale	the scaling factor for the data, the slope at the transition point in the s-curve transform. Must be greater than 0.
shape	a value that determines how quickly the curve approaches the limits of -1 or 1. Must be greater than 0.

Details

The basic equation for the s-curve is $z/(1 + \text{abs}(z)\text{^shape})\text{^}(1/\text{shape})$, where z is scale*(x-location).

The function sCurve computes the forward transform and the function IsCurve computes the inverse [sCurve] transform, or back-transform.

Value

A numeric vector of the transformed or back-transformed values in x.

Note

The sCurve function is related to the hyperbolic function in that both can represent mixing models for flow in stream water chemistry. The sCurve function is more flexible when there are distinct upper and lower limits to the concentration. The hyperbolic function is more flexible for open-ended concentrations for either high or low flows. Also, sCurve would typically use log-transformed values for flow.

See Also

hyperbolic

Examples

```
## Not run:
# Basic changes to the s-curve
curve(sCurve(x), -5,5, ylim=c(-1,1))
# Shift to left
curve(sCurve(x, location=1), -5,5, add=TRUE, col="red")
# increase slope
curve(sCurve(x, scale=2), -5,5, add=TRUE, col="cyan")
# increase rate
curve(sCurve(x, shape=2), -5,5, add=TRUE, col="purple")

## End(Not run)
```

---

seasons *Seasonal Categories*

Description

Create categories for any definitions of seasons by month and day.

Usage

```
seasons(x, breaks, Names = paste("Season Ending ", breaks, sep = ""))
```

Arguments

x	any vector of valid dates or date-time data of class "Date" or "POSIXt."
breaks	either month names of the end of the seasons or specific days in the form of "mm/dd," where mm is the 2-digit month and dd is the 2-digit day. Breaks in the form of "mm/dd" indicate the last day of each season. Breaks must be in calendar order.
Names	optional names for the seasons.

Details

The default names for the seasons are of the form "Season Ending ...," where ... is derived from breaks.

Value

A factor of seasonal categories.

See Also

month (in lubridate package)

Examples

```
## Just two seasons
seasons(as.Date(c("2001-03-31", "2001-06-30", "2001-09-30")), breaks=c("June", "December"))
## The equivalent using mm/dd format
seasons(as.Date(c("2001-03-31", "2001-06-30", "2001-09-30")), breaks=c("06/30", "12/31"))
## Not run:
# Apply to a real dataset
library(smwrData)
data(QW05078470)
transform(QW05078470, Seas=seasons(DATES, breaks=c("June", "December")))

## End(Not run)
```

---

seqCollapse *Collapse a Sequence*

Description

Collapse a numeric sequence into a compact form that represents continuous ranges and discontinuous values.

Usage

```
seqCollapse(x, sequential = "-", skips = ", ")
```

Arguments

x	an integer vector, missing values and repeated values are permitted and removed before collapsing.
sequential	the separator for sequential values.
skips	the separator for gaps in the sequence.

Value

A character string that represents that data in x in a compact form. If x is empty, then "" is returned.

Note

This function is commonly used to express years in a compact form.

See Also

paste (in base package)

Examples

```
# A single value
seqCollapse(1968)
# A single, continuous range of values
seqCollapse(1968:1992)
# A collection of continuous and individual values
seqCollapse(c(1968:1992, 1998, 2002, 2006:2012))
```

setFileType *File or Object Name*

Description

Create a new file or object name from an old name with an optional new suffix.

Usage

```
setFileType(filename, type = "tmp", replace = FALSE)
```

Arguments

filename	a single character string of the name of the file or object.
type	character string identifying the new suffix.
replace	logical (TRUE or FALSE): replace the current suffix?

Value

A character string like filename but with a new suffix.

Note

This function is designed as a support function for many functions.

Author(s)

Dave Lorenz, original coding by Jim Slack, U.S. Geological Survey retired.

Examples

```
# Replace the .dat suffix with .txt
setFileType("TestName.dat", "txt", replace=TRUE)
```

setTZ *Set Time Zone*

Description

Set the time-zone information for dates and times.

Usage

```
setTZ(x, TZ, force.stz = FALSE)
```

Arguments

x	the date-time data, generally class "POSIXct."
TZ	time-zone code or time-zone name, see **Details**.
force.stz	force standard time specified in TZ. Useful for Arizona times, where daylight savings is not used, or in other cases where all times are recorded as standard time. Also useful when the dates and times are recorded over the transition from daylight savings time to standard time. Valid only in the United States. Used only when retrieving data from a single time zone.

Details

The time-zone information should be a standard name like those described in http://en.wikipedia.org/wiki/List_of_zoneinfo_time_zones. For the convenience of users in the United States, correct conversion is provided for the time-zone codes of EST, EDT, CST, CDT, MST, MDT, PST, PDT, AKST, AST, AKDT, ADT, HAST, and HST. However, time data in States like Arizona, where savings time is never used, would use time-zone information specified like " America/Phoenix" to avoid the possibility of setting savings time when it is not appropriate.

Value

Data like x, but with times adjusted by the time-zone information.

Note

The time-zone information is a characterisitic of the data and not of each individual value. If the data in x come from different time zones, then a time zone is selected from the data and used as the base—the dates in x are correctly converted to the selected time zone and a warning is issued.

See Also

as.POSIXct (in base package)

Examples

```
TestDts <- as.POSIXct(c("2010-05-28 09:50:00", "2010-11-29 15:20:00"))
setTZ(TestDts, c("PDT", "PST"))
# Try setting to different time zones
setTZ(TestDts, c("PDT", "CST"))
```

---

shiftData *Shift Data*

Description

Returns a vector like the input, but with the position of the data shifted up or down.

Usage

```
shiftData(x, k = 1, fill = NA, circular = FALSE)
```

Arguments

x	any vector.
k	a positive or negative whole number of positions to shift the data. Positive values shift data to a higher position and negative values shift data to a lower position.
fill	a scalar value like x used to fill in the first k positions or the last -k positions if circular=FALSE. Ignored if circular=TRUE. The default value is NA. If x is class "factor," then fill must be NA or a valid level in x.
circular	logical (TRUE or FALSE). If TRUE, then treat x as a circular buffer, rotating values from the end into the beginning if k is positive and vice versa if k is negative. If FALSE, then use the value of fill. The default value is FALSE.

Value

A vector like x, with data shifted in position.

See Also

lag (in stats package)

Examples

```
shiftData(1:5, k=1)
# [1] NA  1  2  3  4
shiftData(1:5, k=1, circ=TRUE)
# [1] 5 1 2 3 4
```

---

show-methods *Methods for Function* show *for Time-of-Day Objects*

Description

Display the time of day.

Usage

```
## S4 method for signature 'timeDay'
show(object)
```

Arguments

object	the object to be printed.

Value

The object is printed and returned invisibly.

`sumComposition` *Percentage Composition*

Description

Compute percentage or proportion of elements in a composition.

Usage

```
sumComposition(x, ..., Range = 100)
```

Arguments

`x`	any numeric vector, matrix, or data frame containing only numeric columns.
`Range`	the output range, generally 100 for percentages (the default) or 1 for proportions.
`...`	any additional vectors or matrices.

Details

Missing values are permitted in `x` or `...` and result in missing values for the row in the output.

Value

A matrix with columns matching all of the data in `x` and `...{}` with rows summing to `Range`.

Note

This function is designed to meet a very simple need in some applications like constructing data for Piper (Piper, 1944) or trilinear diagrams. For more in-depth manipulations of compositional data, the user is directed to the `compositions` or other similar package.

References

Piper, A.M., 1944, A graphical procedure in the geochemical interpretation of water analyses: Transactions of the American Geophysical Union, v. 25, p. 914-923.

Examples

```
# Create tiny dataset
TinyCations <- data.frame(Ca=c(32, 47, 28), Mg=c(10,12,15), Na=c(7, 5, 7))
sumComposition(TinyCations)
```

timeDay-class *Time of Day*

Description

Class "timeDay" describes the time of day without any reference to the date. The data are stored as seconds since midnight.

Slots

`time` the time of day in seconds since midnight.

`format` a character string indicating the format to display the time of day.

Objects from the Class

Objects can be created by calls of the form `as.timeDay(time, format)`.

Examples

```
showClass("timeDay")
```

untable *Contingency Table*

Description

Constructs a data frame from the count data in a contingency table, using the column and row names as classes.

Usage

```
untable(x, rows = "Rows", cols = "Columns", counts = FALSE)
```

Arguments

`x`	a contingency table. Missing values are not permitted.
`rows`	a character string indicating the name of the column containing the data for the rows. The default column name is "Rows."
`cols`	a character string indicating the name of the column containing the data for the columns. The default column name is "Columns."
`counts`	a logical value indicating whether there should be one row in the result for each observation, which is the default `counts = FALSE`, or whether there should be a column that contains the number of counts for each row and column class, `counts = TRUE`.

Value

A data frame containing two columns named from `rows` and `cols` and an optional column named "Counts" if `counts` is set to `TRUE`.

Note

The output for this function can be used for input to contingency table analysis functions that require a data frame rather than a contingency table. To convert a column from factor to ordered, use the ordered function.

See Also

ordered (in base package)

Examples

```
## Create a small synthetic data matrix
mdat <- matrix(seq(6), nrow = 2, ncol=3,
   dimnames = list(c("row1", "row2"), c("C.1", "C.2", "C.3")))
untable(mdat)
```

---

waterYear *Water Year*

Description

Create an ordered factor or numeric values from a vector of dates based on the water year.

Usage

```
waterYear(x, numeric = FALSE)
```

Arguments

x	an object of class "Date" or "POSIXt." Missing values are permitted and result in corresponding missing values in the output.
numeric	a logical value that indicates whether the returned values should be numeric TRUE or an ordered factor FALSE. The default value is FALSE.

Value

An ordered factor or numeric vector corresponding to the water year.

Note

The water year is defined as the period from October 1 to September 30. The water year is designated by the calendar year in which it ends. Thus, the year ending September 30, 1999, is the "1999 water year."

See Also

year (in lubridate package)

Examples

```
library(smwrData)
data(QW05078470)
## Return an ordered factor
waterYear(QW05078470$DATES)
```

whichRowCol *Identify Rows and Columns*

Description

Identifies the row and column numbers (indexes) of TRUE values in a logical matrix.

Usage

```
whichRowCol(x, which = "both")
```

Arguments

x	a logical matrix. Missing values are treated as FALSE
which	a character string indicating what should be returned.

Value

A matrix of the row and column number if which is "both." Otherwise a named vector of the row number, if which is "row," or column number, if which is "col." Only the first character is needed.

Note

Some comparisons, %in% for example, will return a vector rather than a matrix and cause whichRowCol to fail.

See Also

row, col, which (all in base package)

Examples

```
# Simple case to find a single value
whichRowCol(matrix(1:20, ncol=4) == 16)
# Where are the missing values in a data set?
library(smwrData)
data(MenomineeMajorIons)
whichRowCol(sapply(MenomineeMajorIons, is.na))
```

[.timeDay *Extract Parts of an Object*

Description

Extract elements of a time-of-day object.

Usage

```
## S3 method for class 'timeDay'
x[i]
```

Arguments

x	the object.
i	an index specifying elements to extract. See Extract for details.

Value

The subset of x indicated by i.

See Also

Extract (in base package)

%cn% *Partial Value Matching*

Description

Matches partial values, such as substrings.

Usage

```
x %cn% pattern
```

Arguments

x	the character vector to be matched. Missing values are permitted.
pattern	the pattern to be matched against, may be a regular expression.

Value

A vector the same length as x of logical values indicating whether pattern is found in the element of x or not.

See Also

%in%, regexpr (both in base package)

Examples

```
## A simple example
c("abc", "def") %cn% "c"
```

Index